Pythonによる脳波パワースペクトル解析 with Python 3.6

The 70th JSA Meeting, June 1, 2023. Teiji Sawa, MD,PhD. Kyoto Prefectural University of Medicine.

1. セボフルラン全身麻酔中の脳波パワースペクトル解析.

実際にセボフルラン全身麻酔中にEEG Analyzerを用いて，BISモニタから取得したデジタル脳波データファイルeeg_bis.tsvを用いて，Pythonによるスペクトル解析を実践する．ここでは，numpy, pandas,matplotlibに加えて，scipyライブラリのfft()関数，signal()関数を利用する.

In [1]:
```
#コード4-1
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.fftpack import fft

#SciPyパッケージからsignal関数インポート
from scipy import signal as sig
```

In [2]:
```
#コード4-2
df_tsv = pd.read_table('eeg_bis.tsv', encoding='utf-8')
```

取り込んだデータの先頭5件を表示する.

In [3]:
```
#コード4-3
df_tsv.head(3)
```

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
0	ch1:	09:18:52	0.00	0.0	0.0	0.00	0.00	0.0	0.00	0.0	0.0	0.00	0.(
1	ch1:	09:18:52	0.00	0.0	0.0	0.00	0.00	0.0	0.00	0.0	0.0	0.00	0.(
2	ch1:	09:18:52	27.35	26.9	27.0	23.05	16.15	9.7	6.35	8.3	13.3	24.65	37.4

取り込んだデータのラスト5件を表示する.

In [4]:
```
#コード4-4
df_tsv.tail(3)
```

Out[4]:

	Ch	Time	ch[0]	ch[1]	ch[2]	ch[3]	ch[4]	ch[5]	ch[6]	ch[7]	ch[8]	ch[9]
109609	ch1:	13:13:31	5.70	5.60	5.65	5.65	5.60	5.6	5.65	5.85	5.65	5.8
109610	ch1:	13:13:31	5.90	5.85	5.90	5.95	6.00	6.0	5.90	6.00	5.95	5.7
109611	ch1:	13:13:31	5.85	5.95	6.15	6.05	6.05	6.2	6.00	5.85	5.85	5.8

データ構造は一行（1ch:）あたり時刻（Time）と, ch[0]からch[15]の16個のμVデータがTabで仕切られて，8行/秒=128データ/秒=128Hzで構成される．pandasのDataFrameであるdf_eegを新たに作成し，コラム名"Time"と"eeg"を設定する．

In [5]:

```
#コード4-5
cols = ['Time', 'eeg']
df_eeg = pd.DataFrame(data=None, index=[], columns=cols)
```

ここでは，df_tsv（109,611行）のうち，深い麻酔状態にある部分（80,000～81,024）の脳波μVデータ1,024行 x 16個をdf_eegに取り込む．

In [6]:

```
#コード4-6
#1,024行のデータだけ取り込む．8行/秒なので，128秒＝2分8秒 data数＝16,384
for row in range(80000, 81024):

#このrange調整で脳波を取り出せる部位が決まる．
  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)
```

df_eegに取り込んだ1,024行x16=16,383件, 128秒のμVデータを表示する．

In [7]:

```
#コード4-7
df_eeg
```

Out[7]:

	Time	eeg
0	12:10:08	21.05
1	12:10:08	17.45
2	12:10:08	14.20
3	12:10:08	7.45
4	12:10:08	-0.25
...	...	...
16379	12:12:21	21.85
16380	12:12:21	11.85
16381	12:12:21	9.00
16382	12:12:21	14.05
16383	12:12:21	10.60

16384 rows × 2 columns

df_eegに取り込んだデータをcsvファイルとして保存しておく.

In [8]:
```
#コード4-8
df_eeg.to_csv("eeg_linealized.csv")
```

df_eegの時間とμVデータをそれぞれ新たな配列Timeとeegに移す.

In [9]:
```
#コード4-9
#データフレームの脳波データを配置
time = df_eeg['Time'].values
eeg = df_eeg['eeg'].values
```

配列eegに取り込んだデータをplot()で表示する (図4-1).

In [87]:
```
#コード4-10
#脳波データを時系列グラフ表示 (図4-1)
plt.subplots(figsize=(4,1))
plt.plot(eeg)
```

Out[87]:
```
[<matplotlib.lines.Line2D at 0x7ff7a2a85390>]
```

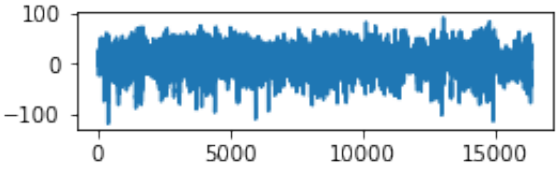


図4-1解説：df_eegの1,024行x16=16,383件, 128秒のμVデータの表示.

2. データ前処理

eegの平均値と標準偏差を算定する.

In [11]:
```
#コード4-11
m=np.mean(eeg)
sd=np.std(eeg)
m, sd
```

Out[11]: (6.136767578124999, 24.727327769075664)

null(NA)データ数をisnull()関数でカウントして合計をsum()関数で求める.

In [12]:
```
#コード4-12
df_eeg['eeg'].isnull().sum()
```

Out[12]: 0

外れ値の除去：df_eegのコピーを作成, コピーに対して, 平均値から標準偏差の3倍をはみ出るデータをNoneに変更する.

In [13]:
```
#コード4-13
df_eeg_copy1 = df_eeg.copy()
cnt_a = 0
cnt_b = 0

for i in range(len(eeg)):
  if df_eeg_copy1['eeg'][i] > m+3*sd:
    df_eeg_copy1['eeg'][i] = None
    cnt_a += 1

for i in range(len(eeg)):
  if df_eeg_copy1['eeg'][i] < m-3*sd:
    df_eeg_copy1['eeg'][i] = None
    cnt_b += 1

    cnt = cnt_a + cnt_b
cnt_a, cnt_b, cnt
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ip
ykernel_launcher.py:8: 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

/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ip
ykernel_launcher.py:13: 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
  del sys.path[0]
```

Out[13]: (5, 147, 152)

外れ値を除去したあと，再度null(NA)データ数をisnull()関数でカウントして，合計をsum()関数で求める.

In [14]:
```
#コード4-14
df_eeg_copy1['eeg'].isnull().sum()
```

Out[14]: 152

df_eeg_copy1のコピーを作成，コピーに対してnull(NA値)を前後の値でinterpolate()関数でスプライン補間する.

In [15]:
```
#コード4-15
df_eeg_copy2 = df_eeg_copy1.copy()
df_eeg_copy2.interpolate(inplace=True)
```

df_eegのコピーのnull(NA)データ数をisnull()関数でカウントして，合計をsum()関数で求める.

In [16]:
```
#コード4-16
df_eeg_copy2['eeg'].isnull().sum()
```

Out[16]: 0

外れ値を前後のデータでスプライン補間修正したファイルをto_csv()関数でcsv保存.

In [17]:
```
#コード4-17
df_eeg_copy2.to_csv("eeg_linealized_ip.csv")
```

eegデータを配列eeg_ipへコピーする.

In [18]:
```
#コード4-18
eeg_ip = df_eeg_copy2['eeg'].values
```

eeg_ipデータをplot()で表示する.

In [88]:
```
#コード4-19 (図4-2)
#欠損値を補完補正した脳波データを時系列グラフ表示
plt.subplots(figsize=(4,1))
plt.plot(eeg_ip)
```

Out[88]:
```
[<matplotlib.lines.Line2D at 0x7ff7c01495f8>]
```

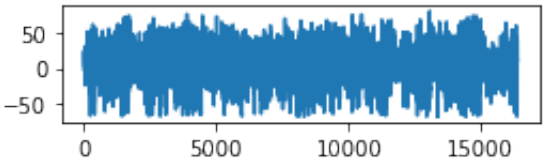


図4-2解説：前処理されたdf_eegのμVデータ（16,383点, 128秒）の表示.

eeg_ipデータから，512件（4秒, eeg_512）と，8,192件(64秒, eeg_8192)を取り出す.

In [20]:
```
#コード4-20
eeg_512 = eeg_ip[1800:2312]
time_512 = time[1800:2312]
eeg_8192 = eeg_ip[1800:9992]
time_8192 = time[1800:9992]
```

eeg_512をグラフ表示する.

In [21]:
```
#コード4-21 (図4-3)
#欠損値を補完補正した脳波データを時系列グラフ表示
plt.subplots(figsize=(4,1))
plt.ylim(-100, 100)
plt.plot(eeg_512, linewidth =0.5)
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x7ff7a1413c50>]
```

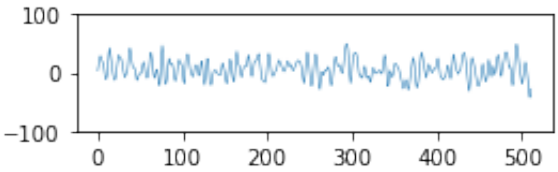


図4-3解説：前処理されたeeg_512のμVデータ（512点, 4秒）の表示.

eeg_8192のデータをグラフ表示する.

In [22]:
```
#コード4-22 (図4-4)
#欠損値を補完補正した脳波データを時系列グラフ表示
plt.subplots(figsize=(4,1))
plt.ylim(-100, 100)
plt.plot(eeg_8192, linewidth =0.5)
```

Out[22]: [<matplotlib.lines.Line2D at 0x7ff7e0b7d3c8>]

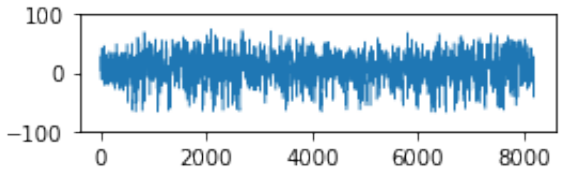


図4-4解説：前処理されたeeg_8192のμVデータ（8,192点, 64秒）の表示.

3. 脳波の窓関数処理パワースペクトル解析

FFT解析のパラメータをセット． ここではサンプリング周波数128とする.

In [23]:
```
#コード4-23
#FFT関数のパラメータセット
N=len(eeg_512) #総データポイント数
N2=len(eeg_8192) #総データポイント数
f_s = 128 #サンプリング周波数
T = 1.0/f_s #データ幅(秒)
s_rate = 1/f_s #サンプリングレート
```

デシベル変換式を関数定義する.

In [24]:
```
#コード4-24
#dB変換
def db(x, dBref):
  y = 20 * np.log10(x / dBref) #変換式
  return y
#FFT関数の中身の演算を確認
1.0/(2.0*T), N/2
```

Out[24]: (64.0, 256.0)

scipyのfft()関数でeeg_512(4秒分）データをFFT解析する.

In [25]:
```
#コード4-25
xfft = np.linspace(0, int(1.0/(2.0*T)), int(N/2))
yfft=fft(eeg_512)
```

単純なFFT解析でのパワースペクトルをナイキスト数N/2を周波数上限として描く (図4-5)． （■プログラミング豆知識4-1：FFT解析処理と規格化定数）

In [26]:
```
#コード4-26 (図4-5)
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
plt.plot(xfft, 2.0/N*abs(yfft[0:int(N/2)]), linewidth =0.5, \
         label="periodogram")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', \
         borderaxespad=1, fontsize=12)
plt.grid()
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.yscale("log")
```

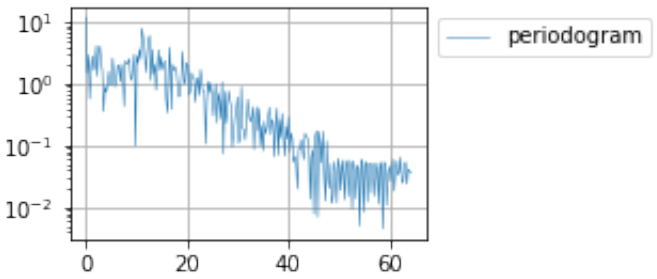


図4-5解説：eeg_512 (128Hz, 512点, 4秒）に対する単純なFFT解析(ピリオドグラム）によるPSD. 対数表示.

デシベル変換する.

In [27]:
```
#コード4-27
dBref = 1
yfft_db = db(2.0/N*abs(yfft[0:int(N/2)]), dBref)
```

単純なFFT（ピリオドグラム）によるPSDをデジベル変換表示する (図4-6).

In [28]:
```
#コード4-28 (図4-6)
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
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.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

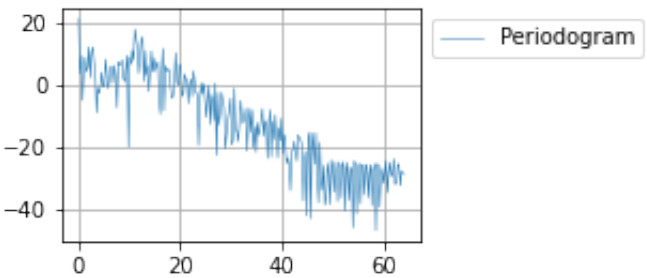


図4-6解説：eeg_512 (128Hz, 512点, 4秒）に対する単純なFFT解析(ピリオドグラム）によるPSD. デシベル表示.

scipyのsignalモジュールを利用してHann窓を作成する.

In [29]:
```
#コード4-29
#窓関数アプライを準備 scipyで.
han = sig.hann(N) #ハニング窓の作成
acf_han = 1/(sum(han)/N) #Amplitude Correction Factorを計算
print("acf_han: ", acf_han)
```

```
acf_han:  2.0039138943248536
```

eeg_512にHann窓を適応して，FFT演算を行う.

In [30]:
```
#コード4-30
#窓関数補正
#dataは時間波形, FFT,正規化,振幅補正を実施
fft_data_acf_han = acf_han*np.abs(fft(eeg_512*han)/(N/2))
```

Hann窓関数を適応したピリオドグラムによるPSDを表示する (図4-7).

In [31]:
```
#コード4-31 (図4-7)
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
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.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

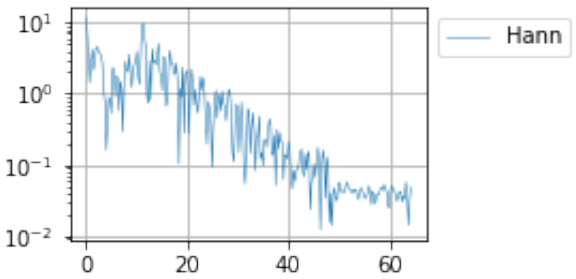


図4-7解説：eeg_512 (128Hz, 512点, 4秒）に対するHann窓関数処理を加えたFFT解析によるPSD. 対数表示.

デシベル変換する (図4-8).

In [32]:
```
#コード4-32
dBref = 1
yfft_acf_db_han = db(fft_data_acf_han[0:int(N/2)], dBref)
```

Hann窓関数を適応したピリオドグラムによるPSDをデシベル表示する.

In [33]:
```
#コード4-33
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
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.ylim(-50,25)
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

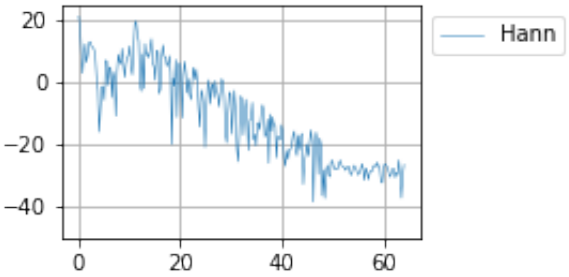


図4-8解説：eeg_512 (128Hz, 512点, 4秒）に対するHan窓関数処理を加えたFFT解析による PSD. デシベル表示.

単純なピリオドグラムとHann窓の適応とのPSDを比較する (図4-9).

In [34]:
```
#脳波パワースペクトル表示 (図4-9)
plt.subplots(figsize=(3,2))
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.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

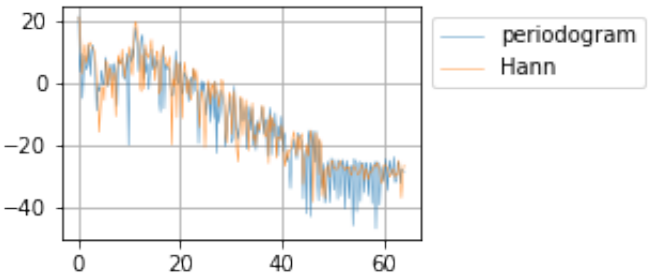


図4-9解説：eeg_512 (128Hz, 512点, 4秒）に対する単純なピリオドグラムとHan窓関数処理を加えたFFT解析によるPSD比較. デシベル表示.

Hann窓，Hamming窓，Blackman窓を作成する.

In [35]:

```
#コード4-35
#複数n窓関数を用意　今回は，numpyで.
window_n = np.hanning(N)
window_m = np.hamming(N)
window_b = np.blackman(N)
x = np.arange(0,N,1) #横軸

#窓関数の補正値
acf_bm=1/(sum(window_b)/N)

#blackman窓：FFT後の数値に掛ければOK
F_abs_amp_bm = acf_bm*np.abs(fft(eeg_512*window_b)/(N/2))
58
dBref = 1
yfft_acf_db_bm = db(F_abs_amp_bm[0:int(N/2)], dBref)
```

Blackman窓関数をアプライしたPSDをHann窓と比較するグラフ表示する (図4-10).

In [36]:

```
#コード4-36 (図4-10)
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
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.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.ylim(-50,25)
```

Out[36]: (-50.0, 25.0)

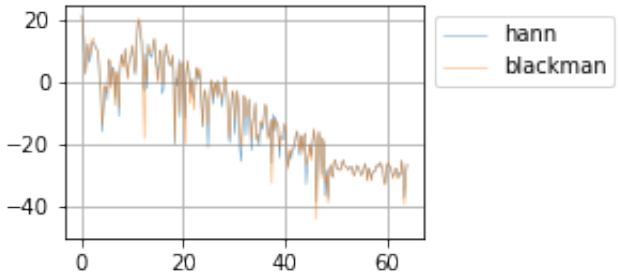


図4-10解説：eeg_512 (128Hz, 512点, 4秒）に対するBlackman窓関数処理PSDをHann窓処理PSDと比較する． デシベル表示.

Blackman窓関数処理は，Hann窓関数と比較して，脳の解析の臨床には意義の低い50Hz以上の高周波領域のパワー振幅を抑制したが，全体として大きな差は無い.

プログラミング豆知識4-1：FFT解析処理と規格化定数 fft()関数を用いた離散フーリエ変換では，y = fft(x)/(N/2)というようにN/2で除している．N/2の部分はFFT解析における規格化定数と呼ばれる．FFT処理したデータの大きさを元の信号と対応させために，FFT 処理データにN/2で除する（2/Nを乗じる）ことが必要である. 2倍する理由はナイキスト周波数を中心とした対称な周波数の波の成分について補正をしていることからである.

Pythonによる脳波スペクトログラム解析

1. 脳波複数解析セグメントのパワースペクトルの実践

窓関数の選択ができるようにする．Hann窓を選択する．解析のデータ窓は512で，ウインドウ間は50%オーバーラップとする.

In [37]:

```
#コード5-1
#サンプリングレート
dt = 1/128
#分析枠データの全体データ比率
split = 512/N2
#オーバーラップ率
overlap = 0.5
#窓関数選択: hanning, hamming, blackman
window = "hanning"

#窓の用意
# ハニング窓
if window == "hanning":
  window_select = "hanning"
  print("Hanningを選択")
# ハミング窓
elif window == "hamming":
  window_select = "hamming"
  print("Hammingを選択")
# ブラックマン窓
elif window == "blackman":
  window_select = "blackman"
  print("Blackmanを選択")
# ハニング窓
else:
  window_select = "hanning"
  print("Hanningを選択")
```

```
Hanningを選択
```

脳波記録を解析セグメントごとにスプリットして解析できるようにする関数data_split()を定義する．ここではeeg_8192の8,192点のデータが512点の解析窓毎にオーバーラップ率0.5(つまり256点ずつシフト）して連続的にFFT解析を行っていく．eeg_8192に対しては8,192/256-1=31セグメントに分割されることとなる．これらのセグメントに対して効率よくFFT解析を行い，グラフ化するために以下の関数群を作成しておく．

In [38]:

```
#コード5-2
# データをFFT解析幅にスプリットする関数で,
# スプリットされたデータを配列として返す.
def data_split(_t, _x, _split, _overlap):
  split_data = []
  #1フレームサンプル数
  one_frame_cnt = int(len(_t)*_split)
  #オーバーラップサンプル数
  overlap_cnt = int(one_frame_cnt*_overlap)
  print("フレームサンプル数: ", one_frame_cnt)
  print("オーバーラップサンプル数: ", 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)
```

次にFFT解析用関数を定義する.

In [39]:
```
#コード5-3
#FFT関数
def FFT(_data_input, _dt, _window_select):
    data_cnt_adj = len(_data_input[0])
    #窓関数処理
    #Hanning
    if _window_select == "hanning":
        window_select = np.hanning(data_cnt_adj)
    #Hamming
    elif _window_select == "hamming":
        window_select = np.hamming(data_cnt_adj)
    #Blackman
    elif _window_select == "blackman":
        window_select = np.blackman(data_cnt_adj)
    #Hanning
    else:
        window_select  = np.hanning(data_cnt_adj)
    #窓関数後信号
    x_windowed = _data_input[1]*window_select
    #FFT計算
    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)
    #窓関数補正
    acf=1/(sum(window_select)/data_cnt_adj)
    F_abs_amp = acf*F_abs_amp
    #ナイキスト定数まで抽出
    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]
```

FFT解析結果をプロットする関数plot_TTF()を定義する.

In [40]:
```
#コード5-4
#FFTの結果をグラフ表示（平均値の表示に使用）
def plot_FFT(_t, _x, _fq, _F_abs_amp):
    fig = plt.figure(figsize=(6, 2))
    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.yscale("log")
    plt.plot(_fq, _F_abs_amp, linewidth =0.5)
```

スプリットされた脳波を表示させる関数plot_split_eeg()を定義する.

In [41]:
```
#コード5-5
#スプリットされた脳波を描写
def plot_split_eeg(_split_data, \
                   _FFT_result_list, \
                   _frame_cnt, \
                   _FFT_result_cnt):

    #スプリットされた脳波データ
    fig3 = plt.figure(figsize=(5, 4))
    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
```

スプリット脳波を全部まとめて表示させる関数plot_FFT_all()を定義する.

In [42]:
```
#コード5-6
#個々のスプリット脳波データ
def plot_FFT_all(_split_data, \
                 _FFT_result_list, \
                 _frame_cnt, \
                 _FFT_result_cnt, \
                 _overlap_cnt, \
                 _dBref):

    #個々のスプリットデータのスペクトラム
    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
```

データを行列matrixに3次元配置する関数set_3D_data()を定義する.

In [43]:

```
#コード5-7
#3次元データ配置
def set_3D_data(_split_data,
                _FFT_result_list,
                _frame_cnt,
                _FFT_result_cnt,
                _overlap_cnt):

    # 3次元データ：T＝時間, Y＝周波数,
    # z＝パワーをリスト形式の配列に収める.
    #時間軸配列へのデータ配置
    T_pre= np.arange(0, 6, 6/_frame_cnt)
    T=[]
    for i in T_pre:
        T.append([i] *(1+_overlap_cnt))

    print("時間配列2次元データ数：", len(T))
    print("時間配列1次元データ数：", len(T[0]))

    #周波数軸配列へのデータ配置
    Y_pre= _FFT_result_cnt[0].tolist()
    Y=[]
    for i in T_pre:
        Y.append(Y_pre)
    print("周波数配列2次元データ数：", len(Y))
    print("周波数配列1次元データ数：", len(Y[0]))

    #周波数軸配列へのデータ配置
    Y_pre= _FFT_result_cnt[0].tolist()
    Y=[]
    for i in T_pre:
        Y.append(Y_pre)
    print("周波数配列2次元データ数：", len(Y))
    print("周波数配列1次元データ数：", len(Y[0]))

    #周波数パワー軸配列へのデータ配置
    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("周波数パワー配列2次元データ数：", len(Z))
    print("周波数パワー配列2次元データ数：", len(Z[0]))

    #3次元データをnumpy配列へ変換
    T=np.array(T)
    Y=np.array(Y)
    Z=np.array(Z)
    return T, Y, Z
```

64秒間8,192データの時間軸配列を設定する.

In [44]:
```
#コード5-8
Time_adj= np.linspace(0, 64, N2)
```

eeg_8192（64秒分)のデータについて，PSD算定の処理を行う．ここでは31セグメントに分割される．スプリットされた脳波データを表示する (図5-1).

In [45]:
```
#コード5-9
#EEGデータのオーバーラップ分割.
split_data = data_split(Time_adj, eeg_8192, split, overlap)
frame_cnt =len(split_data)
print("総フレーム数: ", frame_cnt)

#FFT演算を行う.
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)

#オーバーラップサンプル数
overlap_cnt = int(one_frame_cnt*overlap)

#パワー平均化
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)
```

```
フレームサンプル数:  512
オーバーラップサンプル数:  256
総フレーム数:  31
```

In [46]:
```
#コード5-10 (図5-1)
#スプリット脳波の描画
plot_split_eeg(split_data, \
               FFT_result_list, \
               frame_cnt, \
               FFT_result_cnt)
```

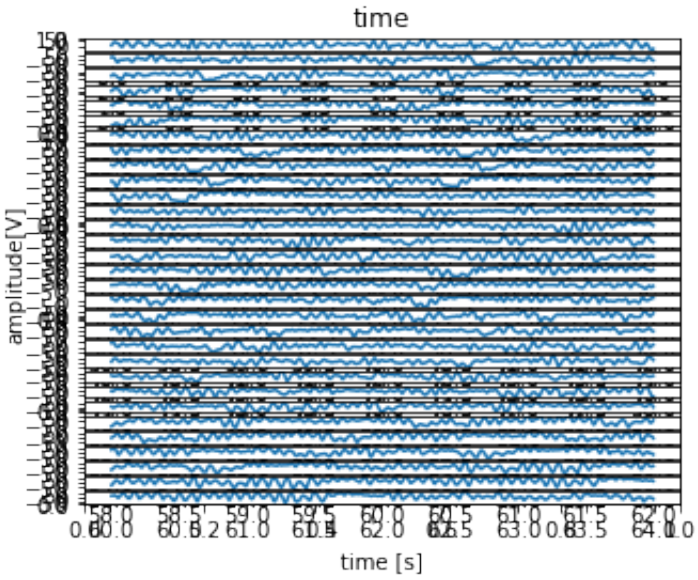


図5-1解説：eeg_8192 (128Hz, 8,192点, 64秒）に対してスプリットした脳波セグメントを表示.

対象脳波波形と全体で平均化されたスペクトルを表示する (図5-2).

In [47]:
```
#コード5-11
#平均化されたスペクトラム
plot_FFT(Time_adj, eeg_8192, \
         fq_ave, F_abs_amp_ave)
```

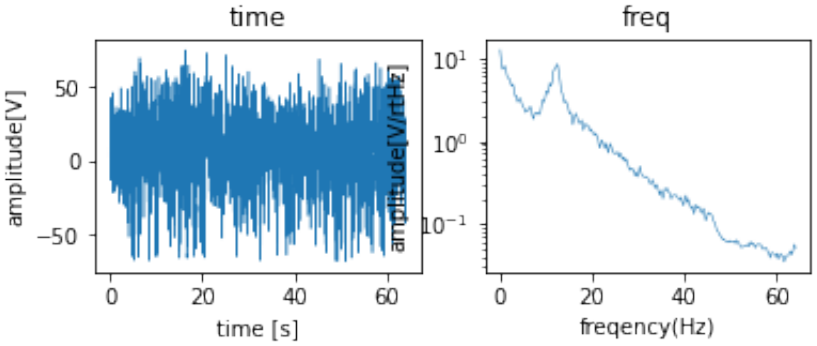


図5-2解説：eeg_8192 (128Hz, 8,192点, 64秒）(左図）に対してスプリットした脳波セグメントに適応したFFT解析より得られたPSDを平均化した(右図).

2. スペクトログラムの構築

解析セグメントに分割された脳波データに対して，FFT解析を行い，その結果得られた3次元情報（時間，周波数，周波数パワー）をスペクトログラムとしてカラーマップ2次元図に構築する．まずはset_3D_data()関数で3Dデータのセット行列を構築する.

In [48]:
```
#コード5-12
#3Dデータのセット：時間，周波数，周波数パワーの2次元配列
#：各次元のデータ数は一致する必要あり．
time2, freq, power = \
set_3D_data(split_data, \
            FFT_result_list, \
            frame_cnt, \
            FFT_result_cnt, \
            overlap_cnt)
```

```
時間配列2次元データ数： 31
時間配列1次元データ数： 257
周波数配列2次元データ数： 31
周波数配列1次元データ数： 257
周波数配列2次元データ数： 31
周波数配列1次元データ数： 257
周波数パワー配列2次元データ数： 31
周波数パワー配列2次元データ数： 257
```

まず始めに，単純なピリオドグラムPSDでスペクトログラムを描く (図5-3).

In [49]:
```
#コード5-13 (図5-3)
nFFT = 512 # the length of the windowing segments
f_s = 128   # the sampling frequency
noverlap=int(nFFT*0.5)
window = np.blackman(nFFT)
plt.figure(figsize=(3, 3))
Pxx, freqs, bins, im = plt.specgram(eeg_8192, \
                                    NFFT=nFFT, \
                                    Fs=f_s, \
                                    scale='dB', \
                                    window=window, \
                                    noverlap=noverlap, \
                                    cmap='jet')
plt.colorbar()
```

Out[49]: <matplotlib.colorbar.Colorbar at 0x7ff7e0d5e978>

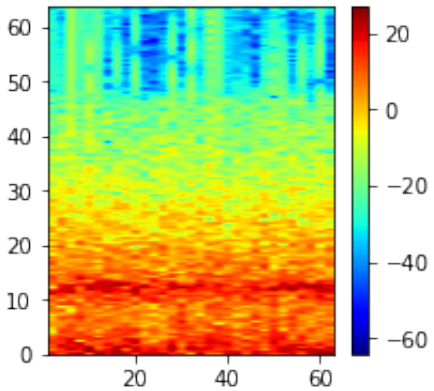


図5-3解説：eeg_8192の8,192点，64秒の脳波データに対してdata_split()関数で31セグメント(オーバーラップ率0.5)に分割された512点ずつに単純なFFT解析が行われて，スペクトログラム表示した.

参考までに，セグメント解析されたPSDを同じグラフ上にまとめて表示させる (図5-4).

In [50]:
```
#コード5-14 (図5-4)
#import matplotlib.pyplot as plt
Bref=1
plt.subplots(figsize=(3,2))
plt.plot(freqs, db(Pxx, Bref), linewidth =0.5)
```

Out[50]:
```
[<matplotlib.lines.Line2D at 0x7ff7f20e70f0>,
 <matplotlib.lines.Line2D at 0x7ff7a15a6d30>,
 <matplotlib.lines.Line2D at 0x7ff7f1f558d0>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55278>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55668>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55e48>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55710>,
 <matplotlib.lines.Line2D at 0x7ff7f1f559e8>,
 <matplotlib.lines.Line2D at 0x7ff7f1f557b8>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55748>,
 <matplotlib.lines.Line2D at 0x7ff7f20e7978>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55160>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55128>,
 <matplotlib.lines.Line2D at 0x7ff7f1f553c8>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55940>,
 <matplotlib.lines.Line2D at 0x7ff7f1f555c0>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55240>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55860>,
 <matplotlib.lines.Line2D at 0x7ff7f1f555f8>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55b00>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55c18>,
 <matplotlib.lines.Line2D at 0x7ff7f1f55da0>,
 <matplotlib.lines.Line2D at 0x7ff7a1590a20>,
 <matplotlib.lines.Line2D at 0x7ff7a1590eb8>,
 <matplotlib.lines.Line2D at 0x7ff7a1590e10>,
 <matplotlib.lines.Line2D at 0x7ff7a1590390>,
 <matplotlib.lines.Line2D at 0x7ff7a1590080>,
 <matplotlib.lines.Line2D at 0x7ff7a1590fd0>,
 <matplotlib.lines.Line2D at 0x7ff7a15902b0>,
 <matplotlib.lines.Line2D at 0x7ff7a1590240>,
 <matplotlib.lines.Line2D at 0x7ff7a1590ef0>]
```

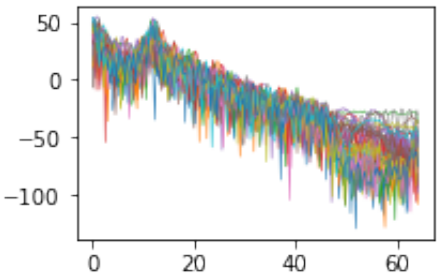


図5-4解説：eeg_8192の8,192点の脳波データに対してdata_split()関数で31セグメント(オーバーラップ率0.5)に分割された512点ずつにFFT解析が行われた． グラフには31本のPSDをまとめて表示 .

次に，より高度なスペクトログラム構築のため，libtfrライブラリ，librosaライブラリ等を導入する．

libtfrを用いたスペクトログラム算定のためのパラメータ定義を行う．そして，続いてlibtfrのtfr_spec()関数を用いた周波数時間再割当て法によるPSDによるスペクトログラム構築ための行列S, およびS2を完成させる．（■ プログラミング豆知識5-1：libtfr）

In [51]:
```
#pip install libtfr  restart kernel!!1
```

In [52]:
```
#コード5-15-1
import libtfr
```

In [53]:
```
#コード5-15-2
from librosa import display
from scipy.signal import chirp
import librosa
```

In [54]:
```
#コード5-16
nfft = 512
f_s = 128
shift = nfft/2
K = 6
tm = 6.0
flock = 1
tlock = 5
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) 215893.53517815363 0.0
(257, 32) 0.0 -100.0
```

specshow()関数にてスペクトログラムを描画する (図5-5).

In [55]:
```
#コード5-17 (図5-5)
plt.subplots(figsize=(3,3))
display.specshow(S2, \
                 x_axis='time', \
                 y_axis='hz', \
                 cmap='jet', \
                 sr=f_s, \
                 hop_length=nfft)
plt.colorbar()
```

Out[55]:
```
<matplotlib.colorbar.Colorbar at 0x7ff7a2453278>
```

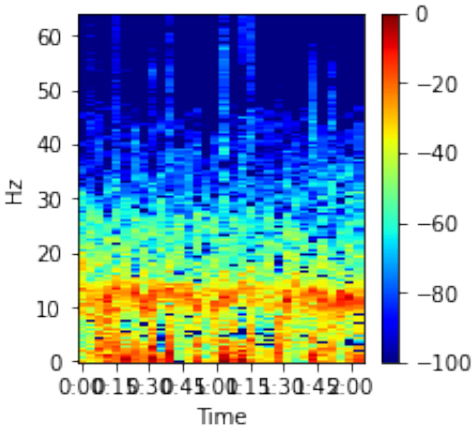


図5-5解説：eeg_8192の8,192点，64秒の脳波データに対してlibtfrのtfr_spec()関数を用いた周波数時間再割当て法によるPSDによるスペクトログラム構築した.

はじめのデータセグメントのPSDを表示させる (図5-6).

In [56]:
```
#コード5-18 (図5-6)
list=[]
for i in range(0, 256):
    list.append(np.mean(S2[i]))
Fq= np.linspace(0, 64, 256)
plt.subplots(figsize=(3,2))
plt.plot(Fq, list, linewidth =0.8)
```

Out[56]:
```
[<matplotlib.lines.Line2D at 0x7ff7a24d02e8>]
```

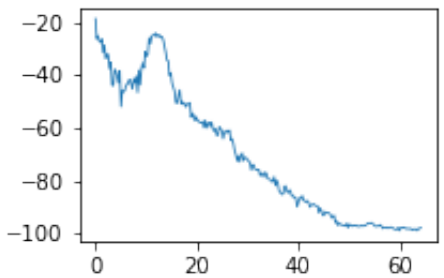


図5-6解説：eeg_8192の8,192点，64秒の脳波データに対して，libtfrのtfr_spec()関数を用いた周波数時間再割当て法によるPSDによるスペクトログラム構築の第1セグメント512点からののPSDを表示.

libtfrのマルチテーパ法mfft_dpss()を用いて，スペクトログラムを作成する (図5-7).

In [57]:
```
#コード5-19
# generate a transform object with size 512 samples
# and 5 tapers for short-time analysis
D = libtfr.mfft_dpss(512, 12, 5)
Z = D.mtstft(eeg_8192, 64)
P = D.mtspec(eeg_8192, 64)
print(np.shape(Z), np.max(Z), np.min(Z))
print(np.shape(P), np.max(P), np.min(P))
```

```
(257, 121, 5) (290.40539223527196+0j) (-284.31933845774296+21.5535222259108
8j)
(257, 121) 50412.71694241948 0.002685783319869248
```

In [58]:
```
#コード5-20 (図5-7)
P = librosa.amplitude_to_db(P, ref=np.max, top_db =100)
plt.subplots(figsize=(3,3))
display.specshow(P, \
                 x_axis='time', \
                 y_axis='hz', \
                 cmap='jet', \
                 sr=f_s, \
                 hop_length = nfft)
plt.colorbar()
```

Out[58]: <matplotlib.colorbar.Colorbar at 0x7ff7a2589c88>

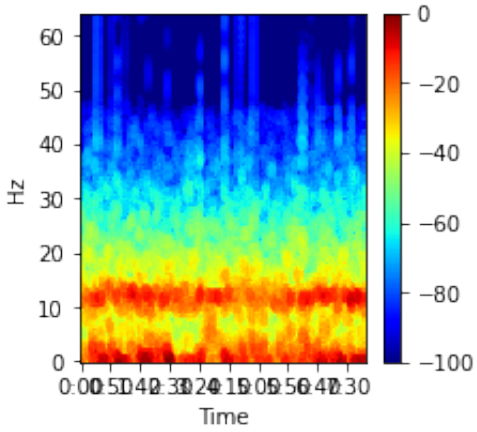


図5-7解説：eeg_8192の8,192点，64秒の脳波データに対して，libtfr.mfft_dpss()よるマルチテーパ法でのスペクトログラム構築した.

マルチテーパ法による第1セグメントのPSDを表示させる (図5-8).

In [59]:
```
#コード5-21 (図5-8)
list3=[]
for i in range(0, 256):
    list3.append(np.mean(P[i]))
plt.subplots(figsize=(3,2))
plt.plot(Fq, list3, linewidth =0.8)
```

Out[59]: [<matplotlib.lines.Line2D at 0x7ff7d11ffe80>]

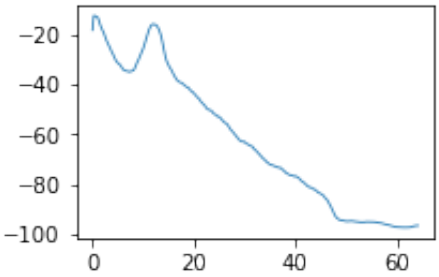


図5-8解説：eeg_8192の8,192点，64秒の脳波データに対して，libtfr.mfft_dpss()よるマルチテーパ法でのスペクトログラム構築の第1セグメント512点からののPSDを表示.

第1セグメントのPSDについて，libtfrの2つの方法（時間再割当て法，マルチテーパ法）の２つを比較する(図5-9).

In [60]:
```
#コード5-22 (図5-9)
fig = plt.figure()
plt.subplots(figsize=(3,2))
plt.plot(Fq, list, linewidth =0.8)
plt.plot(Fq, list3, linewidth =0.8)
```

Out[60]: [<matplotlib.lines.Line2D at 0x7ff7e14902e8>]

<Figure size 432x288 with 0 Axes>

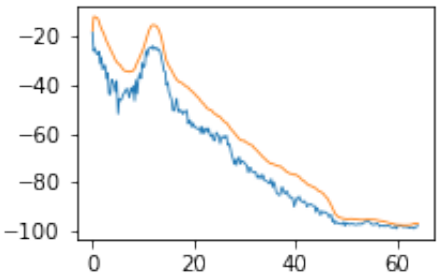


図5-9解説：eeg_8192の8,192点，64秒の脳波データに対して，周波数時間再割当て法，もしくはマルチテーパ法で構築したスペクトログラム第1セグメントPSDを比較表示.

マルチテーパ法では，このように複数のテーパ関数を用いて加重平均することで，スペクトルの漏れやノイズの影響が抑制されたスムーズなPSD図を描くことができる．従って時間軸を加えて3次元データとして描くカラーヒートマップとしてのスペクトログラムにおいても，特徴周波数の輪郭が明瞭な可視化が行われることとなる.

3. スペクトログラムの構築（オプション法）

前節では，matplotlibのspecgram()関数や，librosaのdisplay.specshow()関数を用いて，スペクトログラムのカラーマップを作成した．ここでは，matplotlibのpcolormesh()関数を用いるオプション法を示しておく.

前節のコード5-12 の後，以下のオプションフローを示す．解析セグメントは512データ点，サンプリング周波数128Hz，オーバーラップ率0.5として，numpy.signalのspectrogram()関数をHann窓適応で使用して，スペクトログラム情報を構築するための時系列周波数パワーを変数Sxxに得る．

In [61]:
```
#コード5-23
###option spectrogram
# the length of the windowing segments
nFFT = 512

# the sampling frequency
f_s =128
noverlap=nFFT*0.5
freqs, times, Sxx = sig.spectrogram(eeg_8192, \
                                    fs=f_s, \
                                    window='hanning', \
                                    nperseg=512, \
                                    noverlap=noverlap, \
                                    detrend=False, \
                                    scaling='spectrum')
```

スペクトログラムのカラーマップにはメッシュを描画するmatplotlib.pyplotの pcolormesh()関数を用いる (図5-10)．ただし，カラーマップのレベルを表示するカラーバーの設定には，mpl_toolkits.axes_grid1のmake_axes_locatable()関数を利用して以下のように対処する．

In [62]:
```
#コード5-24 (図5-10)
###option spectrogram
from mpl_toolkits.axes_grid1 import make_axes_locatable
fig, ax = plt.subplots(figsize=(4, 3))
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)
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ipykernel_launcher.py:8: MatplotlibDeprecationWarning: shading='flat' when 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 shading='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
```

Out[62]: <matplotlib.colorbar.Colorbar at 0x7ff7e14d4f28>

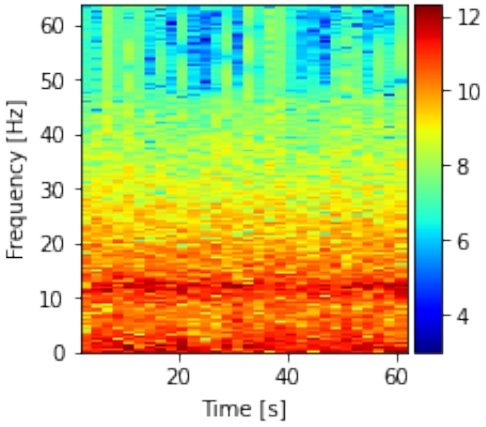


図5-10解説：eeg_8192の8,192点，64秒の脳波データに対してHann窓処理を加えて，pcolormesh()を用いて構築したスペクトログラム.

次に，libtfrのtfr_spec()関数を用いた周波数時間再割当て法により得られた時系列PSD情報S2を用い，matplotlib.pyplotのpcolormesh()関数によるスペクトログラム構築する (図5-11).

In [63]:
```
#コード5-25
###option spectrogram
times_adj = np.append(times, 64)
```

In [64]:
```
#コード5-26
###option spectrogram
fig, ax = plt.subplots(figsize=(4, 3))
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)
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ipykernel_launcher.py:4: MatplotlibDeprecationWarning: shading='flat' when 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 shading='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.
```

Out[64]: <matplotlib.colorbar.Colorbar at 0x7ff7d1286be0>

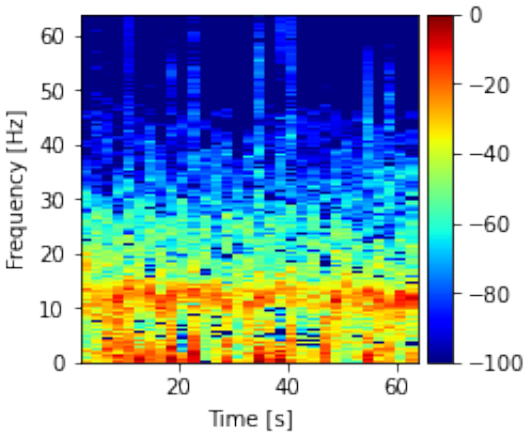


図5-11解説：eeg_8192の8,192点，64秒の脳波データに対してlibtfrのtfr_spec()関数を用いた周波数時間再割当て法によるPSDによるスペクトログラムをpcolormesh()を用いて構築した.

libtfrのマルチテーパ法mfft_dpss()を用いて，スペクトログラムを作成する (図5-12).

In [65]:
```
#コード5-27
###option spectrogram
times_adj2 = np.linspace(0, 64, 121)
```

In [66]:
```
#コード5-28 (図5-12)
###option spectrogram
fig, ax = plt.subplots(figsize=(4, 3))
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]')
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ip
ykernel_launcher.py:4: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specif
y the corners of the quadrilaterals with X and Y, or pass shading='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.
```

Out[66]: `Text(0.5, 0, 'Time [s]')`

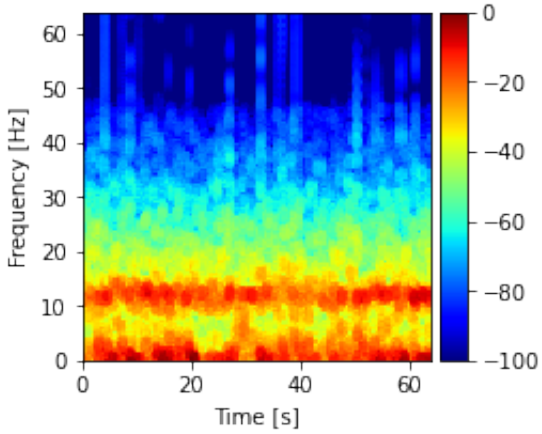


図5-12解説：eeg_8192の8,192点，64秒の脳波データに対して，libtfr.mfft_dpss()よるマルチテーパ法でのスペクトログラムをpcolormesh()を用いて構築した．

■ プログラミング豆知識5-1：libtfr libtfr（https://github.com/melizalab/libtfr）は，マルチテーパ時間周波数再割り当て（TFR）スペクトログラムおよび従来のSTFTを計算するための高性能CおよびPythonライブラリである． このライブラリは，基礎となるFFT解析にFFTWを必要とする．FFTW（http://www.fftw.org）は，離散フーリエ変換（DFT）を計算するためのCサブルーチンライブラリである．Linux，Mac OS X，Windows10環境でlibtfrを使用するためには，予めFFTWをインストールする必要がある．著者はMac OS X環境を用いているが，ソースコード(fftw-3.3.8.tar.gz)をダウンロードして，Xcode開発環境を用いて，コンパイルして導入した．

Pythonによるマルチテーパ法解析の理解

1. マルチテーパ法の理解

この章では，5章で示したマルチテーパ法によるパワースペクトル生成の算定方法ついて，Slepianシークエンスの生成と扱い方を含めて，もう少し詳しく分解して確認しておこう． ライブラリmtspec (Python wrapper for the Fortran 90 Multitaper Spectrum Estimation Library, https://krischer.github.io/mtspec/)を導入する1, 2)． Slepianシークエンスとして知られる離散扁長回転楕円体シーケンス (DPSS: Discrete Prolate Spheroidal Sequences)をdpss()関数を用いて5つを描画する． （■ プログラミング豆知識6-1：Slepian シークエンス）

In [67]:
```
#コード6-1 (図6-1)
# §5からのコードの続きで実行してください.
from mtspec import dpss
tapers, lamb, theta = dpss(N, 2.5, 5)
plt.subplots(figsize=(3,2))
for i in range(5):
    plt.plot(tapers[:, i])
```

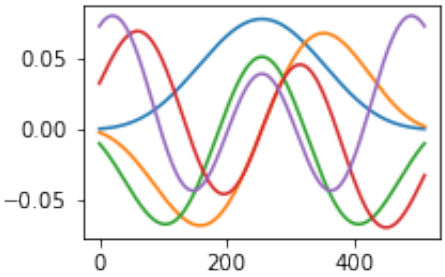
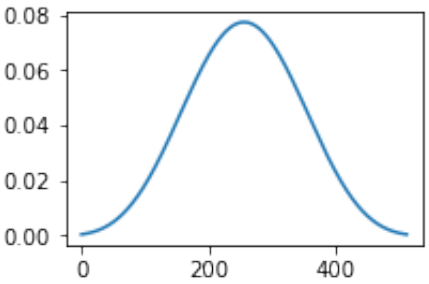


図6-1解説：mtspecモジュールのdpss()関数を用いて，5つのSlepianシークエンステーパ関数(データポイント512)を生成した.

第1テーパ関数を脳波データに乗算して，第1テーパ処理を行う (図6-2:4)．ここでは，テーパ関数のデータ数も解析対象の脳波データ数も512データ数で一致している必要がある.

In [68]:
```
#コード6-2 (図6-2)
#コード6-2_1
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 0])
```

Out[68]:
```
[<matplotlib.lines.Line2D at 0x7ff7e161a7f0>]
```



In [69]:
```
#コード6-2_2 (図6-3)
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 0]*eeg_512)
```

Out[69]: [<matplotlib.lines.Line2D at 0x7ff7a261fba8>]

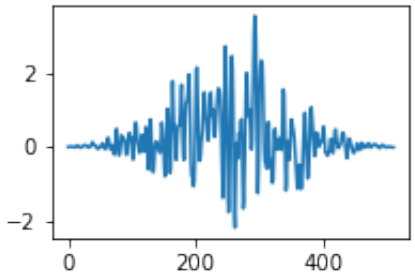


図6-2, 6-3解説：第1テーパ関数とそれをeeg_512に乗じた脳波波形.

In [70]:

```
#コード6-3 (図6-4)
#Amplitude Correction Factorを計算
acf_0 = 1/(sum(tapers[:, 0])/N)
print("acf_0:", acf_0)
#dataは時間波形, FFT,正規化,振幅補正を実施
fft_data_acf_0 = np.abs(fft(tapers[:, 0]*eeg_512)/(N/2))
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
plt.plot(xfft, fft_data_acf_0[0:int(N/2)], \
         linewidth =0.5, \
         label="taper #1")
plt.legend(bbox_to_anchor=(1, 1), \
           loc='upper right', \
           borderaxespad=1, \
           fontsize=12)
plt.yscale("log")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

```
acf_0: 28.843575280051258
```

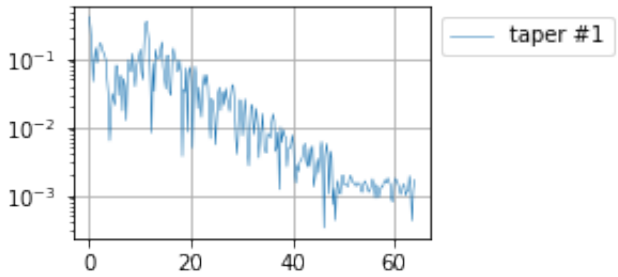


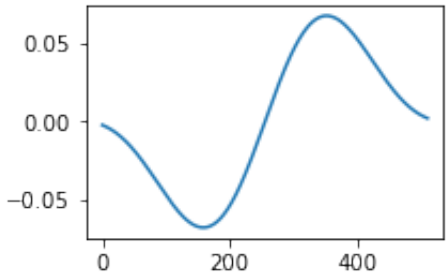
図6-4解説：第1テーパ関数処理した脳波に対するFFT解析によるPSD.

第2テーパ関数を脳波データに乗算して，テーパ処理を行う (図6-5:7).

In [71]:

```
#コード6-4  (図6-5)
#コード6-4_1
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 1])
```

Out[71]: [<matplotlib.lines.Line2D at 0x7ff7d13080f0>]



In [72]:
```
#コード6-4_2 (図6-6)
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 1]*eeg_512)
```

Out[72]: [<matplotlib.lines.Line2D at 0x7ff7a27c8470>]

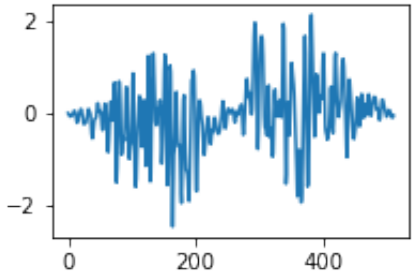


図6-5, 6-6解説：第2テーパ関数とそれをeeg_512に乗じた脳波波形.

In [73]:
```
#コード6-5 (図6-7)
#Amplitude Correction Factorを計算
acf_1 = 1/(sum(tapers[:, 1])/N)
print("acf_1:", acf_1)
#dataは時間波形, FFT,正規化,振幅補正を実施
fft_data_acf_1 = np.abs(fft(tapers[:, 1]*eeg_512)/(N/2))
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
plt.plot(xfft, \
         fft_data_acf_1[0:int(N/2)], \
         linewidth =0.5, \
         label="taper #2")
plt.legend(bbox_to_anchor=(1, 1), \
           loc='upper right', \
           borderaxespad=1, \
           fontsize=12)
plt.yscale("log")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

```
acf_1: -9.266810209712805e+17
```

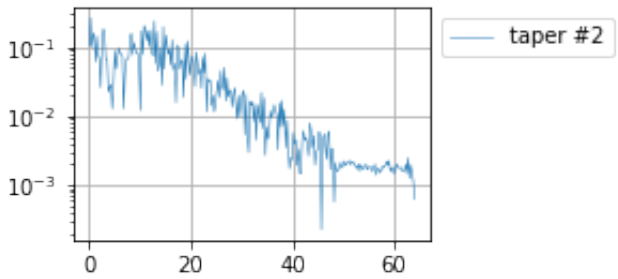
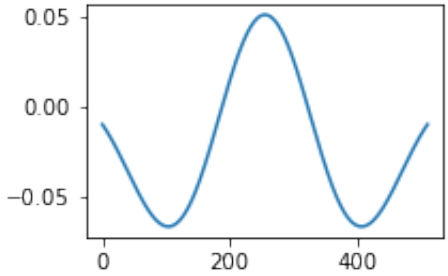


図6-7解説：第2テーパ関数処理した脳波に対するFFT解析によるPSD.

第3テーパ関数を脳波データに乗算して，テーパ処理を行う (図6-8:10).

In [74]:
```
#コード6-6 (図6-8)
#コード6-6_1
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 2])
```

Out[74]:
```
[<matplotlib.lines.Line2D at 0x7ff7a243d0f0>]
```



In [75]:
```
#コード6-6_2 (図6-9)
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 2]*eeg_512)
```

Out[75]:
```
[<matplotlib.lines.Line2D at 0x7ff7f27131d0>]
```

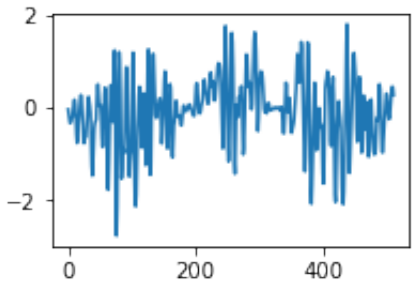


図6-8, 6-9解説：第3テーパ関数とそれをeeg_512に乗じた脳波波形.

In [76]:
```
#コード6-7 (図6-10)
#Amplitude Correction Factorを計算
acf_2 = 1/(sum(tapers[:, 2])/N)
print("acf_2: ", acf_2)
#dataは時間波形, FFT,正規化,振幅補正を実施
fft_data_acf_2 = np.abs(fft(tapers[:, 2]*eeg_512)/(N/2))
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
plt.plot(xfft, \
         fft_data_acf_2[0:int(N/2)], \
         linewidth =0.5, \
         label="taper #3")
plt.legend(bbox_to_anchor=(1, 1), \
           loc='upper right', \
           borderaxespad=1, \
           fontsize=12)
plt.yscale("log")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

```
acf_2:  -43.95616703826029
```

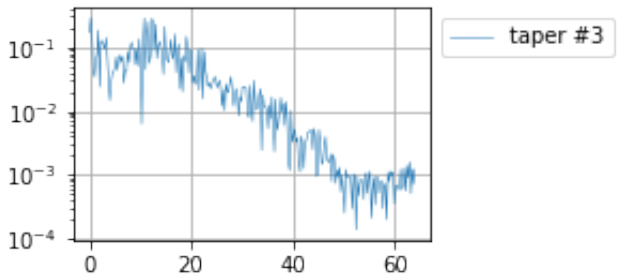
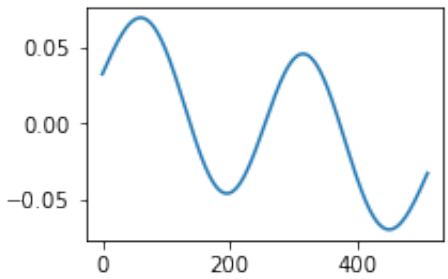


図6-10解説：第3テーパ関数処理した脳波に対するFFT解析によるPSD.

第4テーパ関数を脳波データに乗算して，テーパ処理を行う (図6-11:13).

In [77]:
```
#コード6-8 (図6-11)
#コード6-8_1
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 3])
```

Out[77]:
```
[<matplotlib.lines.Line2D at 0x7ff7a2857d30>]
```



In [78]:
```
#コード6-8_2　(図6-12)
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 3]*eeg_512)
```

Out[78]:
```
[<matplotlib.lines.Line2D at 0x7ff7a287bef0>]
```

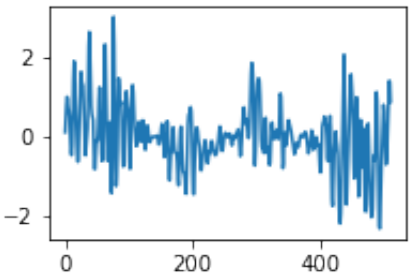


図6-11, 6-12解説：第4テーパ関数とそれをeeg_512に乗じた脳波波形.

In [79]:
```
#コード6-9 (図6-13)

#Amplitude Correction Factorを計算
acf_3 = 1/(sum(tapers[:, 3])/N)
print("acf_3: ", acf_1)

#dataは時間波形, FFT,正規化,振幅補正を実施
fft_data_acf_3 = np.abs(fft(tapers[:, 3]*eeg_512)/(N/2))

#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
plt.plot(xfft, \
         fft_data_acf_3[0:int(N/2)], \
         linewidth =0.5, \
         label="taper #4")
plt.legend(bbox_to_anchor=(1, 1), \
           loc='upper right', \
           borderaxespad=1, \
           fontsize=12)
plt.yscale("log")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

```
acf_3:  -9.266810209712805e+17
```

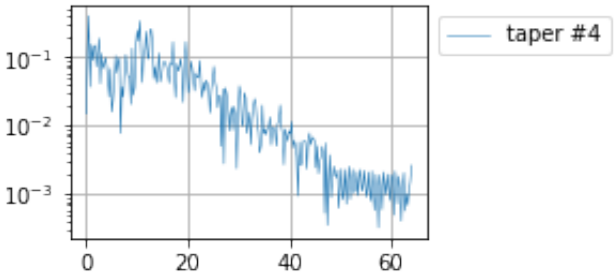
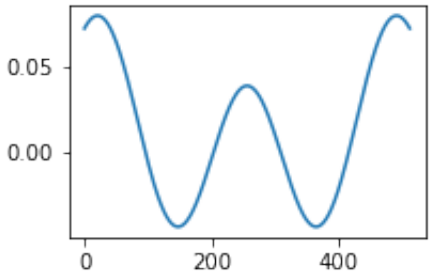


図6-13解説：第4テーパ関数処理した脳波に対するFFT解析によるPSD.

第5テーパ関数を脳波データに乗算して，テーパ処理を行う (図6-14:16).

In [80]:
```
#コード6-10 (図6-14)
#コード6-10_1
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 4])
```

Out[80]:
```
[<matplotlib.lines.Line2D at 0x7ff7d1546080>]
```



In [81]:
```
#コード6-10_2  (図6-15)
plt.subplots(figsize=(3,2))
plt.plot(tapers[:, 4]*eeg_512)
```

Out[81]:
```
[<matplotlib.lines.Line2D at 0x7ff7a290e630>]
```

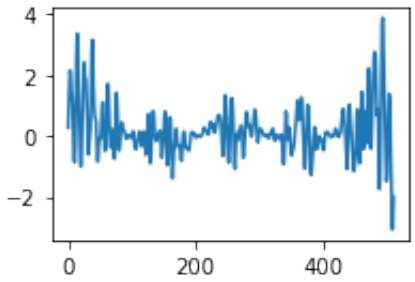
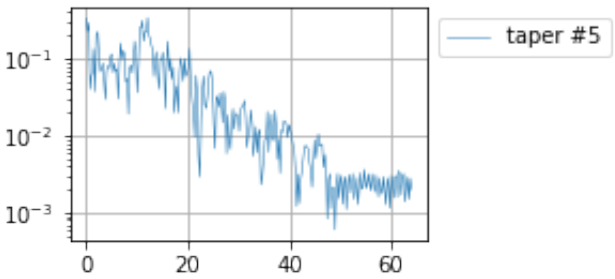


図6-14, 6-15解説：第5テーパ関数とそれをeeg_512に乗じた脳波波形.

In [82]:
```
#コード6-11
#Amplitude Correction Factorを計算
acf_4 = 1/(sum(tapers[:, 4])/N)
print("acf_4: ", acf_4)
#dataは時間波形, FFT,正規化,振幅補正を実施
fft_data_acf_4 = np.abs(fft(tapers[:, 4]*eeg_512)/(N/2))
#脳波パワースペクトル表示
plt.subplots(figsize=(3,2))
plt.plot(xfft, \
         fft_data_acf_4[0:int(N/2)], \
         linewidth =0.5, \
         label="taper #5")
plt.legend(bbox_to_anchor=(1, 1), \
           loc='upper right', \
           borderaxespad=1, \
           fontsize=12)
plt.yscale("log")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
```

```
acf_4:  67.87524658198507
```



In [83]:

```
#コード6-12 (図6-17)
yfft_mt=(fft_data_acf_0\
         +fft_data_acf_1\
         +fft_data_acf_2\
         +fft_data_acf_3\
         +fft_data_acf_4)
dBref = 1/5
yfft_acf_db_mt = db(yfft_mt, dBref)
#脳波パワースペクトル表示:マルチテーパ法PSD
plt.subplots(figsize=(3,2))
plt.plot(xfft, \
         yfft_acf_db_mt[0:int(N/2)], \
         linewidth =0.5, \
         label="multitaper")
plt.legend(bbox_to_anchor=(1, 1), \
           loc='upper right', \
           borderaxespad=1, \
           fontsize=12)
plt.grid()
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.ylim(-50,25)
```

Out[83]:

```
(-50.0, 25.0)
```

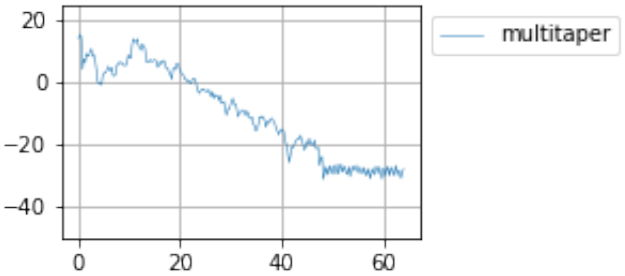


図6-17解説：5つのテーパ処理からFFT演算処理された結果を平均化して求めたPSD.

最後にマルチテーパ法と，ピリオドグラムによるPSDを比較表示する(図6-18).

In [84]:
```
#コード6-13 (図6-18)
#マルチテーパ法PSDと，ピリオドグラムPSDを比較
plt.subplots(figsize=(3,2))
plt.plot(xfft, \
         yfft_db, \
         linewidth =0.5, \
         label="periodogram", \
         alpha=0.8)
plt.plot(xfft, \
         yfft_acf_db_mt[0:int(N/2)], \
         linewidth =0.5, \
         label="multitaper")
plt.legend(bbox_to_anchor=(1, 1), \
           loc='upper right', \
           borderaxespad=1, \
           fontsize=12)
plt.grid()
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.ylim(-50,25)
```

Out[84]: (-50.0, 25.0)

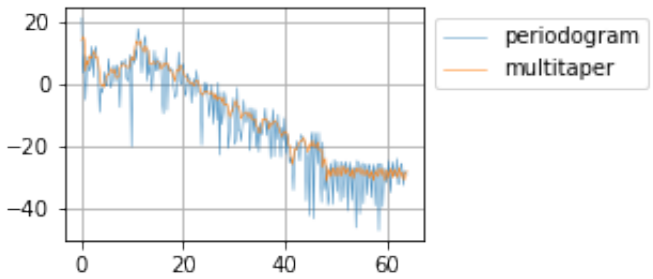


図6-18解説：マルチテーパ法と，ピリオドグラムによるPSDを比較表示させる．

■ プログラミング豆知識6-1：Slepian シークエンス (参考文献: Article-7) 日本語では離散扁長回転楕円体列と呼ばれる． すべての有限エネルギーのシーケンス x[n] のインデックスが設定された [N1,N1+N2] に制限されている場合，次の比率が最大になるのはどのシーケンスであるかを考える．

（Fs はサンプルレートで，|W|<Fs/2 ） インデックスが制限されたシーケンスは帯域 [–W,W] において最大比率のエネルギーをもつ．比率を最大化するシーケンスが，第1の離散扁長回転楕円体列，またはスレピアン列となる．第2のスレピアン列は，比率が最大になり，第1のスレピアン列と直交します．第3のスレピアン列は，積分の比率が最大になり，第1と第2の両方のスレピアン列と直交する．このように操作を継続していくことでスレピアン列は帯域制限されたシーケンスの直交集合を形成する．

In [ ]: