Empirical Wavelet Transform -1 : Python 3.8

Maintenance Phase, General Anesthesia with Sevoflurane

Feb 10, 2023 by Teiji Sawa, Department of Anesthesiology, Kyoto Prefectural University of Medicine

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

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

try EEG waveform

In [2]:
```
# df = pd.read_csv('eeg_1024.csv')
df_tsv = pd.read_table('./eeg_bis_20220629_sev_EME.tsv')
```

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

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

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

In [6]:
```
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)
```

In [7]:
```
eeg_tmp
```

Out[7]: 94.8

In [8]:
```
df_eeg
```

Out[8]:

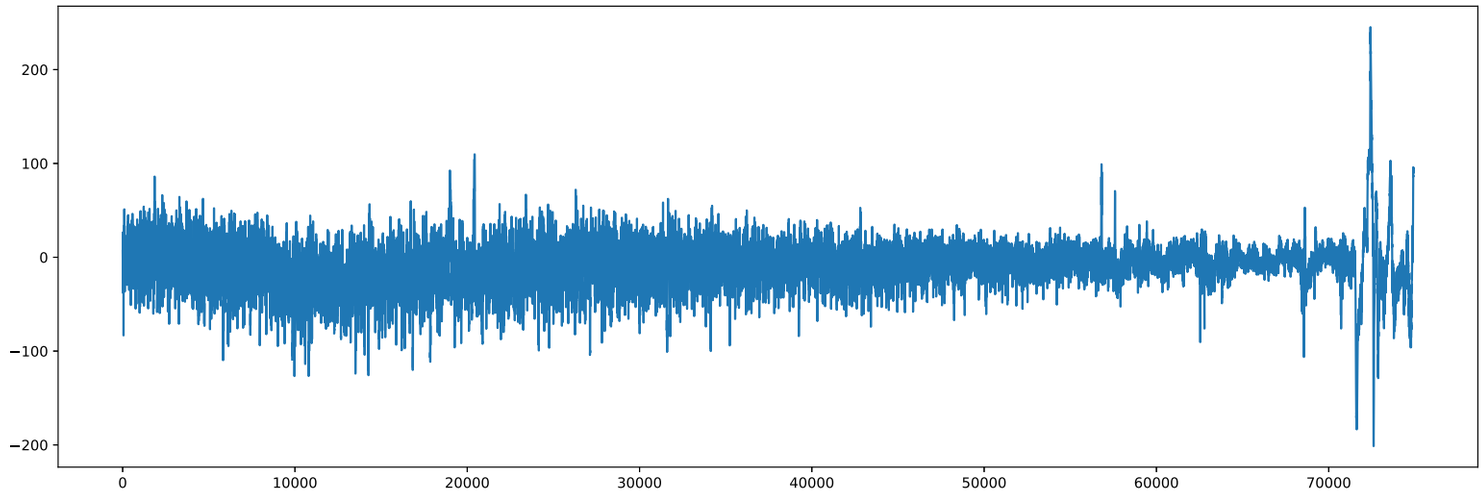
	Time	eeg
0	12:15:17	6.70
1	12:15:17	3.80
2	12:15:17	1.75
3	12:15:17	-1.75
4	12:15:17	-1.95
...	...	...
74907	12:25:18	93.45
74908	12:25:18	95.95
74909	12:25:18	91.35
74910	12:25:18	89.50
74911	12:25:18	94.80

74912 rows × 2 columns

In [9]:
```
# df_eeg.to_csv("data_eeg_ewt2/eeg_linealized_eeg_bis_std2_1.csv")
```

In [10]:
```
# Place brain wave data in the data frame
time = df_eeg['Time'].values
eeg = df_eeg['eeg'].values
```

In [11]:
```
# Time-series graph display of EEG data
fig, ax = plt.subplots(figsize=(18,6))
plt.plot(eeg)
plt.savefig('data_eeg_ewt2/EWT_eeg_1.svg.svg')
```



In [12]:
```
eeg_ip = df_eeg['eeg'].values
```

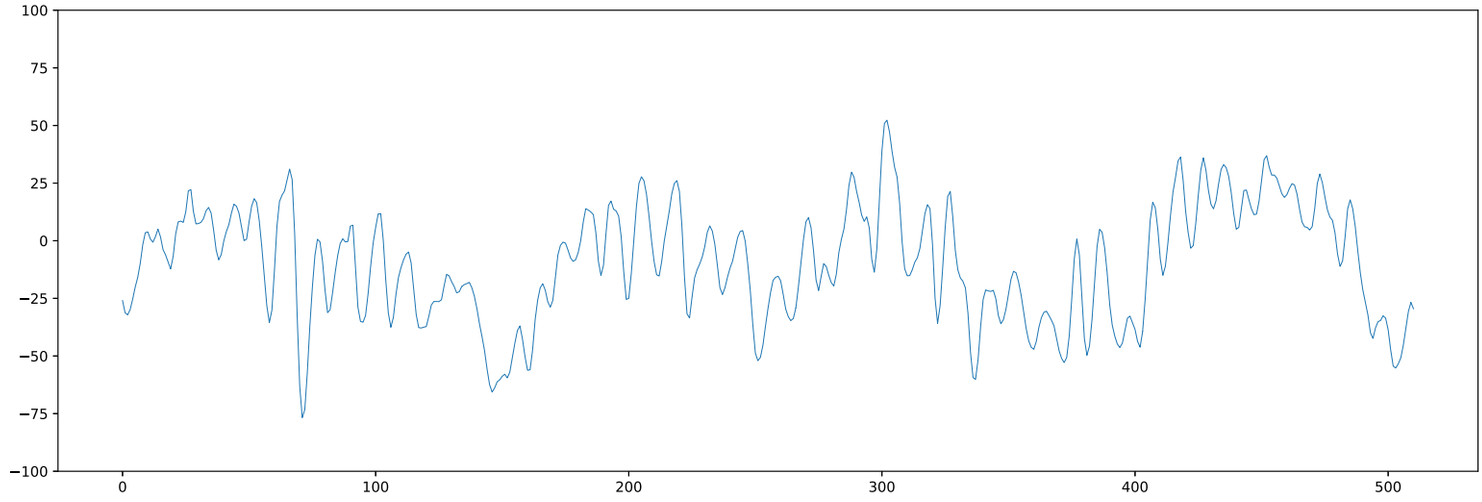
In [13]:
```
# maintenance
eeg_512 = eeg_ip[5000:5512]
time_512 = time[5000:5512]

#transition
#eeg_512 = eeg_ip[60000:60512]
#time_512 = time[60000:60512]

#emergence
#eeg_512 = eeg_ip[66719:67231]
#time_512 = time[66719:67231]
```

In [14]:
```
# 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[0:511], linewidth =0.5)
# plt.savefig('data_eeg_ewt2/EWT_eeg512_2.svg')
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0x7f800b911fd0>]
```

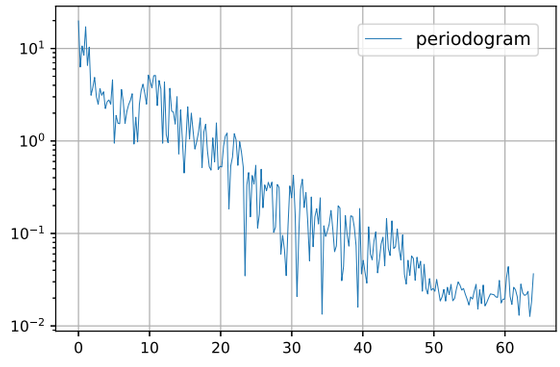


In [15]:
```
N=len(eeg_512[0:511]) # total number of data points
f_s = 128 # sampling frequency
T = 1.0/f_s  # Data width (seconds)
s_rate = 1/f_s # sampling rate
```

In [16]:
```
from scipy.fftpack import fft

xfft = np.linspace(0, int(1.0/(2.0*T)), int(N/2))
yfft=fft(eeg_512[0:511])
```

In [17]:
```
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.yscale("log")
# plt.savefig('data_eeg_ewt2/eeg512_fft_2.svg')
```



Divide into 6 IMFs

In [18]:

```
# Define a dimensional Empirical Wavelet Transform function.
# The input value is f = signal, N = number of waveforms to be separated.
# However, we want the separated waveforms to be narrowband signals separated by N-1 frequency boundaries.

def EWT1D(f, Fs, N):

    # First, the signal is Fourier-transformed with the fft() function
    # to obtain the frequency information of the signal waveform in ff.
    ff = np.fft.fft(f)
    ff = abs(ff[0:int(np.ceil(ff.size/2))])  #Since the numerical value
    # obtained by the Fourier transform is a complex number, an absolute value (amplitude) is obtained.
    # In addition, it is limited to only one side frequency of 0 or more.

    # Determining the boundary value of the Fourier transform power. Find the boundary region N-1 of N waveforms.
    # Simply get the amplitude of each frequency, arrange them in descending order,
    # select the two from the top and put them in the array boundaries as the boundary frequency boundary.
    # If N=3, we want to divide it into 3 waveforms, so there are 2 boundary frequencies,
    # which are stored in boundaries[0] and boundaries[1].

    # Function EWT_Boundaries_Detect() to calculate boundary frequencies
    boundaries = EWT_Boundaries_Detect(ff,N)
    print("boundaries[0]=:", boundaries[0])
    print("boundaries[1]=:", boundaries[1])
    print("boundaries[2]=:", boundaries[2])
    print("boundaries[3]=:", boundaries[3])
    print("boundaries[4]=:", boundaries[4])

    print("Sampling Hz/2=:", round(Fs/2))

    for i in range(len(boundaries)):
        #boundaries[i] = boundaries[i]*np.pi/round(ff.size) # Express the frequency as a number between 0 and π.
        boundaries[i] = boundaries[i]*np.pi/round(Fs/2) # Express the frequency as a number between 0 and π.

    print("ff.size=:", round(ff.size))
    print("boundaries_2[0]=:", boundaries[0])
    print("boundaries_2[1]=:", boundaries[1])
    print("boundaries_2[2]=:", boundaries[2])
    print("boundaries_2[3]=:", boundaries[3])
    print("boundaries_2[4]=:", boundaries[4])

    # Filtering
    # Mirror extension of a signal: A process in which the first and second halves of
    # the original signal waveform are inverted, respectively,
    # and added to the beginning and end of the original signal in order to process the edges of the signal well.
    # As a result, the expanded mirror signal fMirr becomes twice as large as the original signal,
    # and by adding the expanded signal to the head and tail of the original signal,
    # the waveform processing of the stump becomes natural.

    ltemp = int(np.ceil(f.size/2))
    fMirr =  np.append(np.flip(f[0:ltemp-1],axis = 0),f)
    fMirr = np.append(fMirr,np.flip(f[-ltemp-1:-1],axis = 0))

    # Fourier transform the extended mirror signal fMirr and store the frequency data in ffMirr.
    ffMirr = np.fft.fft(fMirr)

    # Create a Meyer filter bank with the EWT_Meyer_FilterBank() function.The filter bank is stored in the array mfb.
    # All you have to do is give the EWT_Meyer_FilterBank() function the separation boundary frequency
    # and the size of ffMirr.
    # For N=3, three filter arrays mfb[0], mfb[1], mfb[2] are stored in the two-dimensional array mfb.
    mfb=EWT_Meyer_FilterBank(boundaries,ffMirr.size)

    # Using the filter set above, the signal is separated into sub-signals.
    # Specifically, the filter set is the Meyer Wavelet, the complex conjugate of the filter
    # in the frequency band and the product over the frequency of the original signal.
    # The frequency components obtained through the filter are converted back into a waveform
    # by inverse Fourier transform.
    # In other words, it is the same process as passing through three bandpass filters in the frequency domain.
    ewt = np.zeros(mfb.shape) # Initialize array ewt with 0
    for k in range(mfb.shape[1]): # Meyer Wavelet transform formula (16-19) in Gilles' EWT original paper
        ewt[:,k] = np.real(np.fft.ifft(np.conjugate(mfb[:,k])*ffMirr)) #N=3なら分離された波形はewt[,0], ewt[,2],ewt[,3]

    ewt = ewt[ltemp-1:-ltemp,:] #H owever, since the original waveform is mirror-extended, t
    # he separated waveform is completed by extracting only the central portion of the actual separated waveform.

    return ewt, mfb, boundaries #Obtain the separation waveform ewt, Meyer Filter Bankmfb,
    # and separation boundary frequency boundaries.
```

In [19]:

```
# The arguments are the function EWT_Boundaries_Detect() that calculates the boundary frequencies,
# the frequency array ff obtained by the Fourier transform of the original waveform,
# and the number N to be separated.
def EWT_Boundaries_Detect(ff,N):

    # regularization
    regFilter = np.ones(10)/10 # Regularization filter: A set of 10 0.1 values.
    # Evaluates by integrating finely divided frequencies in units of 10.
    presig = np.convolve(ff,regFilter,mode = 'same') # Convolve frequency array with regularized filter

    # Boundary extraction
    boundaries = LocalMax(presig,N) # Calculated by N-1 large frequency component division LocalMax() function
    # and stored in boundaries.

    return boundaries+1   # Correct the calculation and return the boundary frequency.
```

In [20]:

```
# Boundary frequency set boundaries are obtained with N-1 large frequency component partitions LocalMax() function.

def LocalMax(ff, N):

    N=N-1
    locmax = np.zeros(ff.size)   # Initialize array with zero
    for i in np.arange(1,ff.size-1):
        if ff[i-1]<ff[i] and ff[i]>ff[i+1]:  # Find extrema (higher than surroundings)
            locmax[i] = ff[i]

    N = min(N,locmax.size)

    maxidxs = np.sort(locmax.argsort()[::-1][:N])  # Use the argosort() function to convert it
    # into an array of array position information, arrange the frequency components in descending order,
    # and store them in maxidxs.

    # middle point between consecutive maxima
    bound = np.zeros(N)
    for i in range(N):  # Find the boundary frequency boud as the average
    #between the centers of adjacent frequency bands.
        if i == 0:
            a = 0
        else:
            a = maxidxs[i-1]
        bound[i] = (a + maxidxs[i])/2
```

```
    return bound  # Returns the boundary frequency boud.
```

In [21]:

```
# Meyer Filter Bank function: Creates a filter set to pass through the frequency band
# based on the boundary frequency information and the number of signal data.

def EWT_Meyer_FilterBank(boundaries,Nsig):

    Npic = len(boundaries)

    # Calculate gamma. Gamma is a coefficient between 0 and 1, the transition coefficient of the boundary region.
    gamma = 1
    for k in range(Npic-1):
        r = (boundaries[k+1]-boundaries[k])/ (boundaries[k+1]+boundaries[k])
        if r < gamma:
            gamma = r
    r = (np.pi - boundaries[Npic-1])/(np.pi + boundaries[Npic-1])
    if r <gamma:
        gamma = r
    gamma = (1-1/Nsig)*gamma

    mfb = np.zeros([Nsig, Npic+1])  # Initialize the filter bank array mfb with zero.

    # Meyer scaling function: Equation (8) from Gilles' EWT original paper
    Mi=int(np.floor(Nsig/2))
    w_pre = np.linspace(0,2*np.pi - 2*np.pi/Nsig,num = Nsig)
    w=np.fft.fftshift(w_pre)
    w[0:Mi]=-2*np.pi+w[0:Mi]
    aw=abs(w)

    yms=np.zeros(Nsig)
    an=1./(2*gamma*boundaries[0])
    pbn=(1.+gamma)*boundaries[0]
    mbn=(1.-gamma)*boundaries[0]

    for k in range(Nsig):
       if aw[k]<=mbn:
           yms[k]=1
       elif ((aw[k]>=mbn) and (aw[k]<=pbn)):
           yms[k]=np.cos(np.pi*EWT_beta(an*(aw[k]-mbn))/2)

    yms=np.fft.ifftshift(yms)

    mfb[:,0] = yms  # mfb[,0] is the first lowpass filter

    for k in range(Npic-1): # Below, EWT_Meyer_Wavelet() function generates Meyer Wavelet from boundary frequency information, gamma, etc.
        mfb[:,k+1] = EWT_Meyer_Wavelet(boundaries[k],boundaries[k+1], gamma, Nsig) # If N=3, the bandpass filter is mfb[,1]

    mfb[:,Npic] = EWT_Meyer_Wavelet(boundaries[Npic-1],np.pi, gamma, Nsig) # If N=3, high pass filter to mfb[,2].

    return mfb # Returns a Meyer Filter Bank.
```

In [22]:

```
# Creating the Beta Function: Equation (7) from Gilles' EWT Original Paper

def EWT_beta(x):

    if x<0:
        bm=0
    elif x>1:
        bm=1
    else:
        bm=(x**4)*(35.-84.*x+70.*(x**2)-20.*(x**3))
    return bm
```

In [23]:

```
# Construction of the Meyer Wavelet Equation (9) in Gilles' EWT original paper

def EWT_Meyer_Wavelet(wn, wm, gamma, Nsig):

    Mi=int(np.floor(Nsig/2))
    w=np.fft.fftshift(np.linspace(0,2*np.pi - 2*np.pi/Nsig,num = Nsig))

    w[0:Mi]=-2*np.pi+w[0:Mi]
    aw=abs(w)

    ymw=np.zeros(Nsig)
    an=1./(2*gamma*wn)
    am=1./(2*gamma*wm)
    pbn=(1.+gamma)*wn
    mbn=(1.-gamma)*wn
    pbm=(1.+gamma)*wm
    mbm=(1.-gamma)*wm

    for k in range(Nsig):
       if ((aw[k]>=pbn) and (aw[k]<=mbm)):
           ymw[k]=1
       elif ((aw[k]>=mbm) and (aw[k]<=pbm)):
           ymw[k]=np.cos(np.pi*EWT_beta(am*(aw[k]-mbm))/2)
           #print("cos=:",ymw[k])
       elif ((aw[k]>=mbn) and (aw[k]<=pbn)):
           ymw[k]=np.sin(np.pi*EWT_beta(an*(aw[k]-mbn))/2)
           #print("sin=:", ymw[k])
    ymw=np.fft.ifftshift(ymw)
    return ymw
```

In [24]:

```
# Next, try to separate the brain wave f2 into three with N=3.
sig = eeg_512[0:512]
ewt2, mfb2, boundaries2 = EWT1D(sig, Fs = 128, N = 6)
```

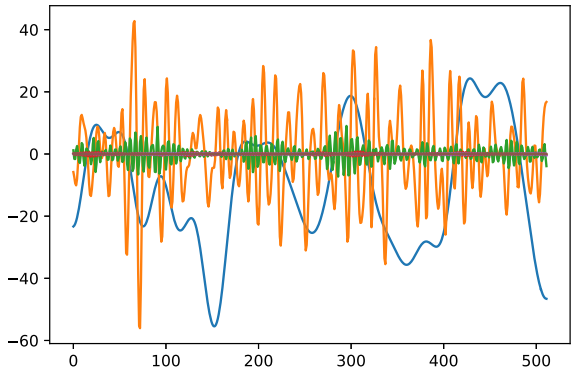
```
boundaries[0]=: 3.5
boundaries[1]=: 20.0
boundaries[2]=: 38.0
boundaries[3]=: 43.5
boundaries[4]=: 46.5
Sampling Hz/2=: 64
ff.size=: 256
boundaries_2[0]=: 0.1718058482431918
boundaries_2[1]=: 0.9817477042468103
boundaries_2[2]=: 1.8653206380689396
boundaries_2[3]=: 2.1353012567368124
boundaries_2[4]=: 2.282563412373834
```

In [25]:

```
# Graph display of separated waveform array ewt
plt.plot(ewt2)
# plt.savefig('data_eeg_ewt2/EWT_imf_1.svg')
```

Out[25]:

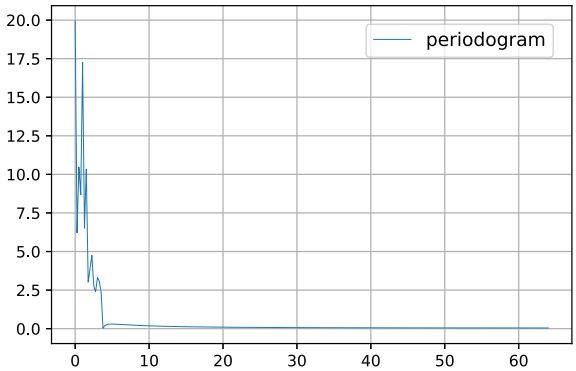
```
[<matplotlib.lines.Line2D at 0x7f7ff8214790>,
 <matplotlib.lines.Line2D at 0x7f7ff8214850>,
 <matplotlib.lines.Line2D at 0x7f7ff8214910>,
 <matplotlib.lines.Line2D at 0x7f7ff82149a0>,
 <matplotlib.lines.Line2D at 0x7f7ff8214ac0>,
 <matplotlib.lines.Line2D at 0x7f7ff8214b50>]
```



In [26]:
```
# Next, acquire actual EEG I 64 data points (128 Hz, 0.5 sec) during sevoflurane anesthesia.
# Let the sampling frequency be 128Hz and the x-axis array be t2.
#T2 = 64
T2 = 512
Fs =128
t2 = np.arange(1,T2+1)/Fs
Ts = 1.0/Fs  #データ幅(秒)
```

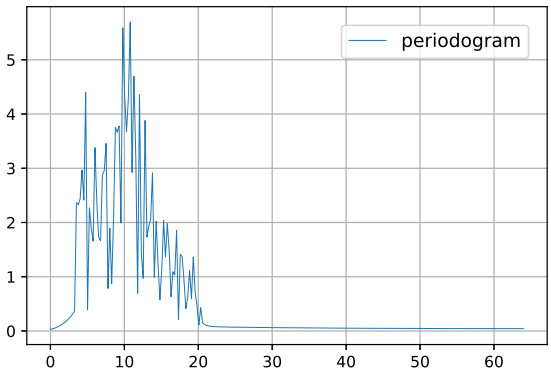
In [27]:
```
xfft_1 = np.linspace(0, int(1.0/(2.0*Ts)), int(T2/2))
yfft_1 =fft(ewt2[:,0])
```

In [28]:
```
plt.plot(xfft_1, 2.0/T2*abs(yfft_1[0:int(T2/2)]), linewidth =0.5,  label="periodogram")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.grid()
# plt.savefig('data_eeg_ewt2/EWT_yfft1_1.svg')
#plt.yscale("log")
```



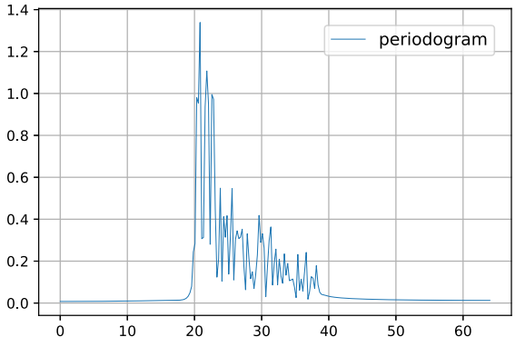
In [29]:
```
xfft_2 = np.linspace(0, int(1.0/(2.0*Ts)), int(T2/2))
yfft_2 =fft(ewt2[:,1])
```

In [30]:
```
plt.plot(xfft_2, 2.0/T2*abs(yfft_2[0:int(T2/2)]), linewidth =0.5,  label="periodogram")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.grid()
# plt.savefig('data_eeg_ewt2/EWT_yfft2_1.svg')
#plt.yscale("log")
```



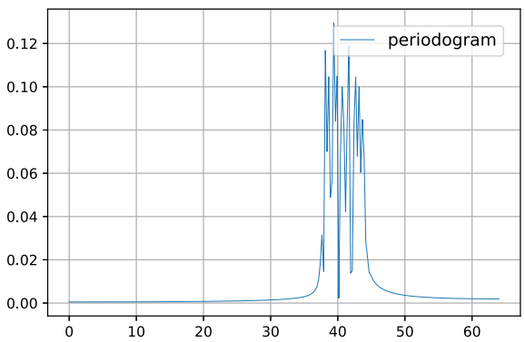
In [31]:
```
xfft_3 = np.linspace(0, int(1.0/(2.0*Ts)), int(T2/2))
yfft_3 =fft(ewt2[:,2])
```

In [32]:
```
plt.plot(xfft_3, 2.0/T2*abs(yfft_3[0:int(T2/2)]), linewidth =0.5,  label="periodogram")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.grid()
# plt.savefig('data_eeg_ewt2/EWT_yfft3_1.svg')
#plt.yscale("log")
```



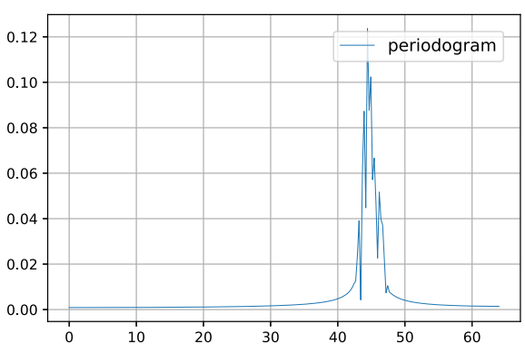
In [33]:
```
xfft_4 = np.linspace(0, int(1.0/(2.0*Ts)), int(T2/2))
yfft_4 =fft(ewt2[:,3])
```

In [34]:
```
plt.plot(xfft_4, 2.0/T2*abs(yfft_4[0:int(T2/2)]), linewidth =0.5,  label="periodogram")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.grid()
# plt.savefig('data_eeg_ewt2/EWT_yfft4_1.svg')
#plt.yscale("log")
```



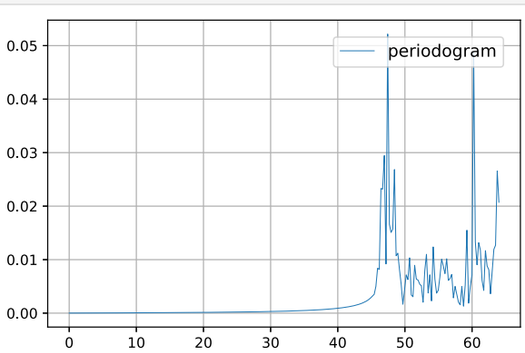
In [35]:
```
xfft_5 = np.linspace(0, int(1.0/(2.0*Ts)), int(T2/2))
yfft_5 =fft(ewt2[:,4])
```

In [36]:
```
plt.plot(xfft_5, 2.0/T2*abs(yfft_5[0:int(T2/2)]), linewidth =0.5,  label="periodogram")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.grid()
# plt.savefig('data_eeg_ewt2/EWT_yfft5_1.svg')
#plt.yscale("log")
```



In [37]:
```
xfft_6 = np.linspace(0, int(1.0/(2.0*Ts)), int(T2/2))
yfft_6 =fft(ewt2[:,5])
```

In [38]:
```
plt.plot(xfft_6, 2.0/T2*abs(yfft_6[0:int(T2/2)]), linewidth =0.5,  label="periodogram")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.grid()
# plt.savefig('data_eeg_ewt2/EWT_yfft6_1.svg')
#plt.yscale("log")
```



Summary¶

1) Empirical Wavelet Transform is to obtain frequency information by first Fourier transforming the original waveform signal. 2) Since we want to separate the waveform into a finite number of multiple waveforms in a narrow frequency band, we divide the whole with a finite number of -1 frequencies at the large frequency component and separate it into each frequency component. For that purpose, first, it is necessary to determine the boundary frequency boundaries where separation is to be performed using the EWT_Boundaries_Detect() function. 3) Next, using the isolated frequencies, if any, to construct a finite set of filters.The filters should have the ability to retain the frequency information in the isolated regions and filter out the rest. In other words, the lowest frequency region has a low-pass filter, the highest frequency region has a high-pass filter, and the region in between has a band-pass filter function. However, the separated waveforms must return to the original waveform when combined, and a smooth transition function is applied as a natural process to the processing of the boundary region of the separated frequency bands. 4) Gamma value and Meyer Wavelet are used for this purpose. Wavelet is a function of small waves as a waveform, but its frequency component is a mountain-like frequency function with a specific frequency band. Multiplying the wavelet frequency band with the signal frequency band is equivalent to applying a bandpass filter.

In [ ]: