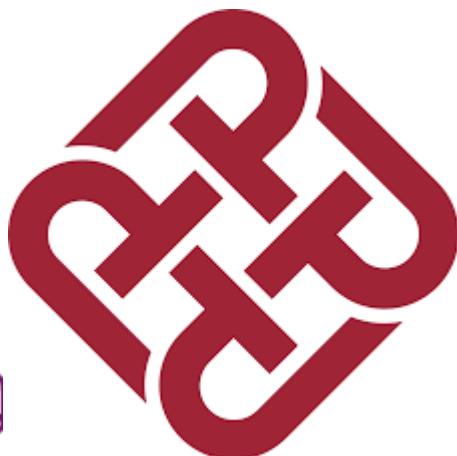


FOOOF-CUHK



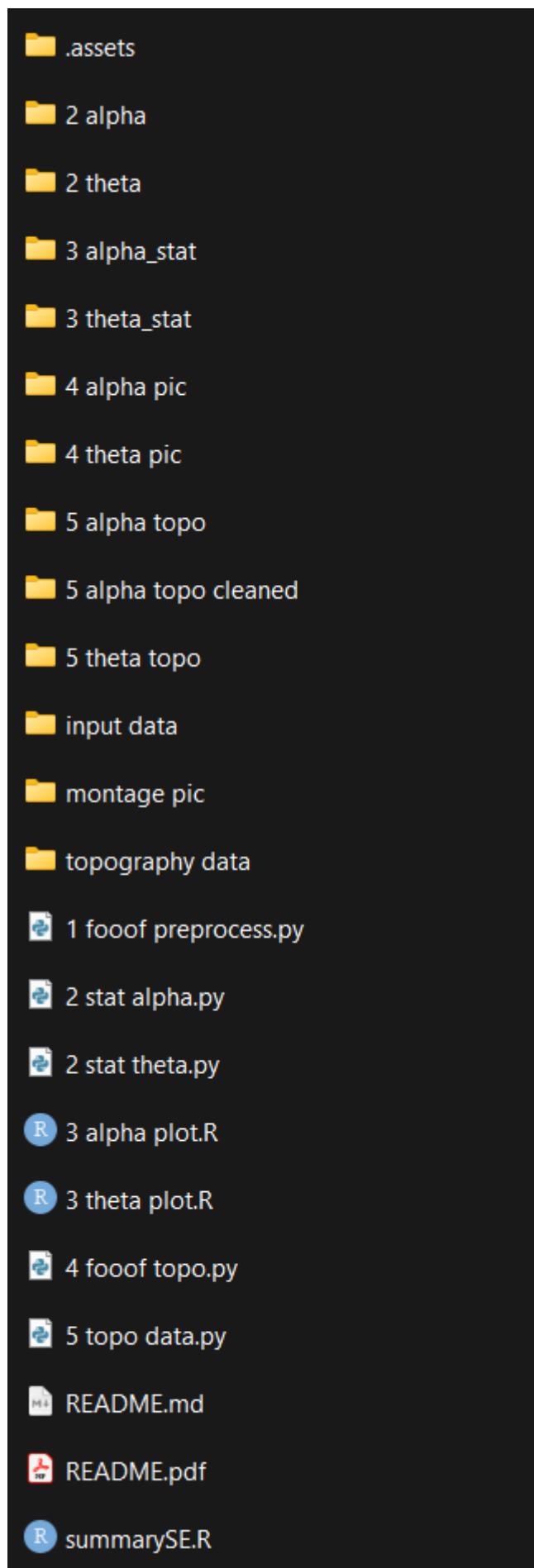
This project encompasses the development and application of the Fitting Oscillations & One Over F ([FOOOF](#)) code within Prof. Urs Maurer's Cognitive Neuroscience Lab at the Chinese University of Hong Kong ([CUHK](#)). The preparation of this document and tutorial is funded by the Hong Kong Polytechnic University ([PolyU](#)) start-up grant (P0043827) awarded to Dr. Shuting Huo. This code includes [FOOOF](#) processing, conducting statistical tests and plotting figures in Python and R. The work is published in Biological Psychology, <https://doi.org/10.1016/j.biopsych.2024.108824>. The code was developed based on the package and tutorial from Thomas Donoghue's team: [FOOOF-fitting oscillations & one over f — fooof-tools.github.io](https://fooof-tools.github.io).

Abbreviation	Full Form
CUHK	The Chinese University of Hong Kong
PolyU	The Hong Kong Polytechnic University
FOOOF	Fitting Oscillations & One Over F
ap	aperiodic component
p	periodic component
CF	central frequency
PW	power
BW	bandwidth
sd	standard deviation
se	standard error

About this file

This file is the guildline for using the code to analyze the provided sample data. The code and documentation for this project were primarily authored by Junior Research Assistant Tak Kwan Lam, and were subsequently reviewed and refined by Research Assistant Professor Dr. Shuting Huo and Junior Research Assistant Yuxi Chen. Since the coder majored in Psychology rather than Computer Science, there is still room for improvement and optimization in this code. This project is only meant to demonstrate the pipeline of FOOOF analysis in Python. We strongly recommend you to read the original work by Thomas for more information and details about [FOOOF](#). Due to the compression in PDF format, please refer to the provided .png files for the best visual experience. This tutorial is written in Typora markdown so it is recommended to use the .md file with Typora markdown reader for the best experience.

Project structure overview



The current project includes fourteen folders, five Python scripts and three R scripts. We can firstly ignore those folders and starts by seeing the codes. The number in the file name indicates the sequence of running and output. Let's start by opening the `1_fooof preprocess.py`.

1 FOOOF preprocess

```
1 # Import required code for visualizing example models
2 import os, shutil
3 # ensure working directory
4 cwd = os.path.dirname(os.path.realpath(__file__))
5 os.chdir(cwd)
6 
7 import pandas as pd
8 import numpy as np
9 from fooof import FOOOF
10 from fooof.plts.annotate import plot_annotated_model
11 import matplotlib.pyplot as plt
12 
13 # Specify the frequency range for alpha range
14 cut_off_component = 15 # the last 15 Hz is not used
15 max_Hz = 40 - cut_off_component
16 freqs = np.array(range(2, 41-cut_off_component))
17 
18 # Specify the output directory
19 dir_path = f'2_alpha/'
20 # Specify the input directory
21 input_dir = 'input data'
22 
23 # Check if dir_path is not empty and it exists
24 if dir_path and os.path.exists(dir_path):
25     # Remove the directory and its contents
26     if os.path.isdir(dir_path):
27         shutil.rmtree(dir_path)
28     else:
29         os.remove(dir_path)
30 # Recreate the directory
31 os.makedirs(dir_path)
32 
33 # create the linear result dataframe to save the results
34 linear_result = pd.DataFrame()
```

We start by specifying the variables and directories for the input and output folders. You may want to refer to other sources (e.g., generative AI) for more details in these codes. As shown above, the raw data was read from the `input data` folder and the output was stored in `2_alpha` and `2_theta` (shown below) folders.

Name	Date modified
EEG10_theta_raw.csv	4/1/2024 1:48 pm
EEG11_theta_raw.csv	4/1/2024 1:48 pm
EEG15_theta_raw.csv	4/1/2024 1:48 pm
EEG16_theta_raw.csv	4/1/2024 1:48 pm
EEG18_theta_raw.csv	4/1/2024 1:48 pm
EEG61_alpha_raw.csv	4/1/2024 1:48 pm
EEG62_alpha_raw.csv	4/1/2024 1:48 pm
EEG67_alpha_raw.csv	4/1/2024 1:48 pm
EEG72_alpha_raw.csv	4/1/2024 1:48 pm
EEG77_alpha_raw.csv	4/1/2024 1:48 pm
EEG78_alpha_raw.csv	4/1/2024 1:48 pm

The above image shows the content in the input data folder, which is the frequency data of each channel.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
2	1	13.616	6.357	2.956	2.635	2.362	2.685	2.329	1.231	1.408	1.776	1.455	0.676	0.4
3	2	16.061	7.383	4.334	3.367	2.783	3.829	5.337	1.963	1.602	2.098	1.761	0.687	0.6
4	3	23.248	7.674	3.934	4.343	4.169	6.316	3.846	4.541	6.173	6.483	2.467	1.089	0.8
5	4	16.143	5.653	4.179	3.843	3.602	6.339	5.319	5.391	7.792	8.805	2.651	1.057	0.8
6	5	26.655	10.544	5.666	3.059	2.037	1.804	1.572	1.29	2.319	2.04	0.999	0.648	0.3
7	6	36.851	13.629	6.39	5.805	3.91	1.797	1.133	1.288	2.314	3.506	1.163	0.415	0.3
8	7	46.068	9.329	4.603	4.84	3.719	4.457	3.737	3.589	4.303	2.226	1.294	0.945	0.7
9	8	29.573	9.919	4.868	4.911	3.689	3.585	3.468	4.396	4.969	2.681	0.979	0.78	0.6
10	9	9.229	5.862	4.737	2.912	2.101	2.537	3.38	1.356	1.378	1.652	1.353	0.977	1.0
11	10	11.481	5.082	4.874	4.445	2.741	2.133	2.276	1.8	1.717	1.891	1.473	1.035	0.9
12	11	27.796	8.677	3.315	4.467	4.38	3.072	4.648	6.665	6.291	2.94	1.28	0.774	0.7
13	12	39.197	7.275	4.6	4.153	4.465	3.804	3.161	5.246	4.68	3.151	1.11	0.858	0.8
14	13	21.52	8.813	3.971	2.427	2.633	1.686	1.077	1.291	1.106	0.693	0.469	0.348	0.3
15	14	15.516	7.446	3.619	2.265	1.591	1.865	1.447	1.248	0.978	0.495	0.379	0.379	0.
16	15	18.089	5.683	3.905	2.322	2.102	1.761	1.685	1.664	1.862	2.678	1.38	0.817	0.6
17	16	22.254	6.761	4.54	1.99	1.502	1.016	1.362	1.623	1.436	2.238	1.306	0.778	0.5
18	17	24.985	8.421	4.909	5.038	4.403	4.853	3.341	3.615	4.696	5.297	1.956	0.884	0.7
19	18	33.874	9.848	6.178	5.324	5.671	4.638	4.021	4.402	4.291	5.442	1.74	0.823	0.
20	19	27.185	6.199	5.135	3.377	1.834	1.503	2.013	2.361	2.839	1.29	0.897	0.859	0.9
21	20	32.126	11.994	5.273	3.39	2.011	2.106	2.828	2.626	2.991	1.695	1.299	0.926	0.8
22	21	22.954	10.589	7.161	4.712	2.783	2.019	1.971	2.015	1.661	2.298	1.791	0.91	0.7
23	22	20.987	9.199	6.103	4.436	2.13	2.368	2.619	2.181	1.875	2.61	2.208	1.52	0.5
24	23	10.683	3.215	3.396	2.192	1.45	1.051	1.035	0.958	0.831	0.881	0.429	0.394	0.3
25	24	9.841	3.251	1.963	1.803	1.059	0.765	0.768	0.682	0.843	0.705	0.353	0.286	0.2
26	25	14.934	2.389	1.102	0.856	1.105	1.004	0.955	1.092	1.981	6.272	7.764	1.722	1.0
27	26	8.507	2.212	1.098	0.872	0.941	1.176	1.053	0.727	1.206	3.241	6.023	2.486	0.7

Inside the .csv file are the voltage (mV) at 1 to 40 Hz frequency (V1 to V40) for every subject. The information for each participant can be found in the last four columns: subject ID (id), experiment conditions (exp, within subject factor), memory load (load, within subject factor), and group (age, adults vs. children, between subject factor).

AN	AO	AP	AQ	AR	AS
V39	V40	id	exp	load	age
0.007	0.005	1108	wmc	1b	0
0.009	0.007	1108	wmc	2b	0
0.034	0.023	1606	wmc	1b	0
0.017	0.012	1606	wmc	2b	0
0.002	0.002	1608	wmc	1b	0
0.007	0.005	1608	wmc	2b	0
0.015	0.015	1612	wmc	1b	0
0.01	0.008	1612	wmc	2b	0
0.004	0.002	1632	wmc	1b	0
0.004	0.002	1632	wmc	2b	0
0.031	0.032	1634	wmc	1b	0
0.057	0.053	1634	wmc	2b	0
0.002	0.001	1635	wmc	1b	0
0.001	0.001	1635	wmc	2b	0
0.01	0.018	1640	wmc	1b	0
0.021	0.011	1640	wmc	2b	0
0.072	0.054	2003	wmc	1b	0
0.051	0.04	2003	wmc	2b	0
0.022	0.014	2004	wmc	1b	0
0.072	0.043	2004	wmc	2b	0
0.02	0.014	2006	wmc	1b	0
0.019	0.01	2006	wmc	2b	0
0.022	0.016	2901	wmc	1b	0
0.009	0.011	2901	wmc	2b	0
0.003	0.001	3001	wmc	1b	1
0.002	0.001	3001	wmc	2b	1

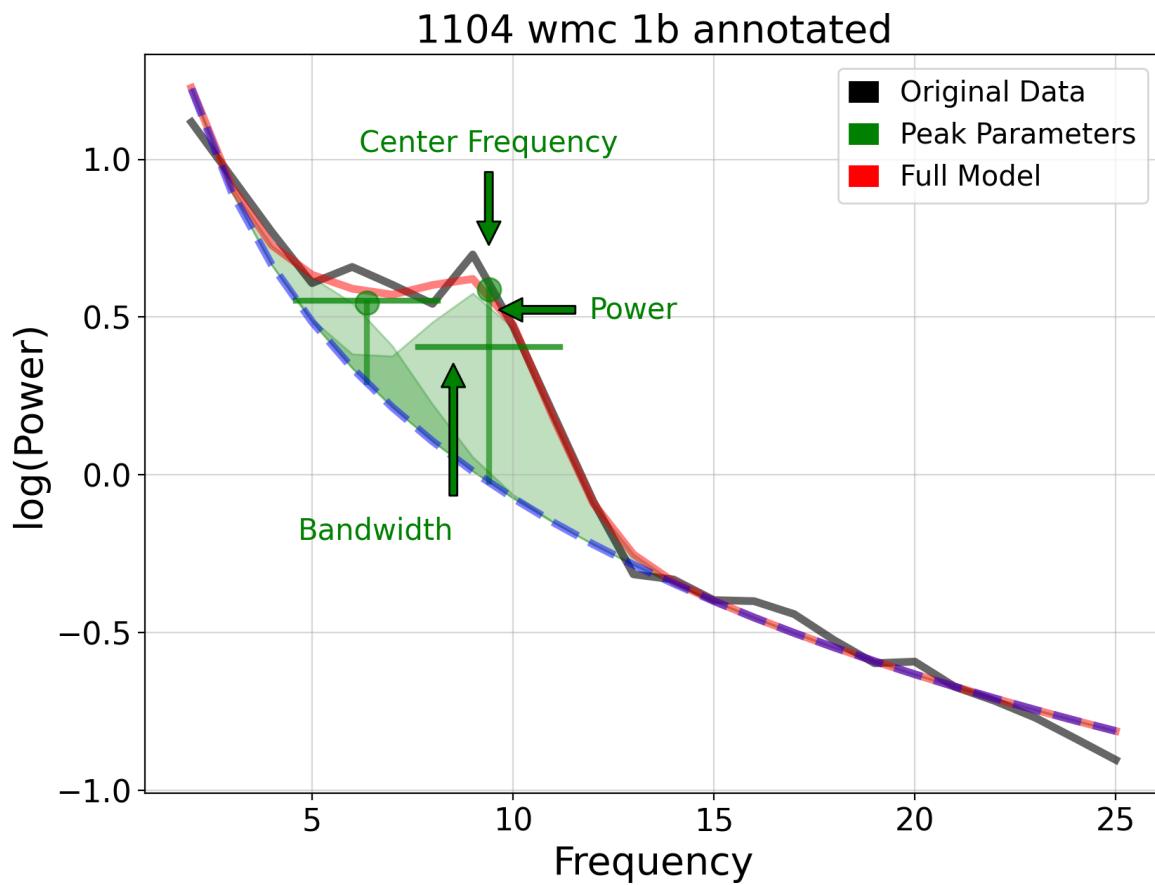
You can also use your own data following the above format and named them by the format

```
EEG[channel num]_[frequency band name]_raw.csv
```

The `1 fooof preprocess` code generates the FOOOF-processed data for each channel and stored in folders corresponding to frequency bands.

Name
1104_wmc_1b_fml anno.png
EEG61_alpha_raw_linear_result.csv
EEG62_alpha_raw_linear_result.csv
EEG67_alpha_raw_linear_result.csv
EEG72_alpha_raw_linear_result.csv
EEG77_alpha_raw_linear_result.csv
EEG78_alpha_raw_linear_result.csv

The `.png` file (shown below) in the 2 alpha folder indicate the function of FOOOF and parameters.



The black curve is the original data read from input data. The blue dashed-curve is the aperiodic (`ap`) component modulated by FOOOF. The red curve is the full model combining the aperiodic (`ap`) and periodic (`p`) components, labeled as the 'Full Model' in the legend. The two gaussian green area is the periodic components. The peak parameters were provided for one of them: center frequency (the center frequency number of this peak), power (the relative power separated from `ap` component of the `p` component), and bandwidth (the frequency range width for this peak). The x-axis is the frequency ranged from 2~25 and y-axis is the logged power.

FOOOFed output

The .csv files in `2 alpha` and `2 theta` folders are the FOOOFed data for each participant in each condition, including `CF` = center frequency, `PW` = log(power), `BW` = bandwidth, `offset` is the intercept for the `ap` component and `exponent` is the slope for `ap` component. `r_squared` is the model r^2 (higher number means better fitting). And `error` is the mean squared error of the model. Noted that there might be multiple peaks for a participant in a condition. Hence, we need to distinguish and filter the data for statistical analysis.

PID	exp	load	age	CF	PW	BW	offset	exponent	r_squared	error	file_name
1104	wmc	1b	0	8.859119	0.621557	3.753233	1.742728	1.858473	0.979716	0.076209	EEG61_alpha_raw
1104	wmc	2b	0	8.628059	0.602743	5.129389	1.819115	1.988517	0.983761	0.074637	EEG61_alpha_raw
1104	wmv	1b	0	6.013343	0.361866	3	1.722888	1.61068	0.988747	0.045797	EEG61_alpha_raw
1104	wmv	1b	0	9.524016	0.406122	3			0.988747	0.045797	EEG61_alpha_raw
1104	wmv	2b	0	8.331516	0.543188	4.698531	1.70127	1.668674	0.97794	0.079324	EEG61_alpha_raw
1108	wmc	1b	0	9.791213	0.689724	3.806853	1.30979	1.539823	0.968018	0.071094	EEG61_alpha_raw
1108	wmc	1b	0	18.499	0.405637	3.945216			0.968018	0.071094	EEG61_alpha_raw
1108	wmc	2b	0	6.596591	0.344977	3	1.513601	1.63149	0.982341	0.055975	EEG61_alpha_raw
1108	wmc	2b	0	10.03163	0.591395	3			0.982341	0.055975	EEG61_alpha_raw
1108	wmc	2b	0	17.77524	0.405478	4.870596			0.982341	0.055975	EEG61_alpha_raw
1108	wmv	1b	0	9.484388	0.554473	4.016173	1.03113	1.331534	0.912957	0.115025	EEG61_alpha_raw
1108	wmv	2b	0	6.623312	0.368704	3	1.255469	1.488449	0.977937	0.062158	EEG61_alpha_raw
1108	wmv	2b	0	9.500049	0.624981	3			0.977937	0.062158	EEG61_alpha_raw
1108	wmv	2b	0	17.4714	0.40756	3.29857			0.977937	0.062158	EEG61_alpha_raw
1605	wmc	1b	0	10.55302	0.744104	4.741311	1.572485	1.503356	0.983843	0.04709	EEG61_alpha_raw
1605	wmc	1b	0	16.2315	0.296549	3.240465			0.983843	0.04709	EEG61_alpha_raw
1605	wmc	2b	0	11.63443	0.659644	5.359043	1.592812	1.597237	0.958709	0.086574	EEG61_alpha_raw
1605	wmv	1b	0	6.553768	0.434507	3	1.557456	1.546957	0.9784	0.067046	EEG61_alpha_raw
1605	wmv	1b	0	10.35763	0.803566	3.000017			0.9784	0.067046	EEG61_alpha_raw
1605	wmv	1b	0	13.26291	0.59977	5.599614			0.9784	0.067046	EEG61_alpha_raw
1605	wmv	2b	0	10.55745	0.701389	4.763212	1.412452	1.582583	0.978599	0.064555	EEG61_alpha_raw
1606	wmc	1b	0	9.055208	1.143976	4.722382	1.499644	1.694526	0.950017	0.13714	EEG61_alpha_raw
1606	wmc	2b	0	9.069774	1.118791	4.10536	1.539636	1.640109	0.960283	0.114135	EEG61_alpha_raw
1606	wmv	1b	0	9.686637	0.952734	4.996314	1.359548	1.581527	0.959097	0.115953	EEG61_alpha_raw
1606	wmv	2b	0	5.918894	0.370217	3	1.486897	1.676932	0.953319	0.121103	EEG61_alpha_raw
1606	wmv	2b	0	9.755661	0.944023	4.01594			0.953319	0.121103	EEG61_alpha_raw
1607	wmc	1b	0	10.03457	0.925415	3	1.566076	1.587583	0.9612	0.096975	EEG61_alpha_raw
1607	wmc	2b	0	9.925508	1.058999	3	1.437959	1.564488	0.940849	0.124519	EEG61_alpha_raw
1607	wmv	1b	0	10.05116	0.890196	3.125528	1.548487	1.612524	0.951814	0.106999	EEG61_alpha_raw
1607	wmv	2b	0	9.783069	0.924741	3	1.610538	1.641304	0.952594	0.107186	EEG61_alpha_raw
1608	1L	0	0.340008	0.001025	0.001210	1.28000	1.74110	0.08040	0.048010	EEG61_alpha_raw

2 stat alpha

The beginning section of `2 stat alpha.py` also specified the variables and directories for input and output folder. `3 alpha_stat` folder stores the statistical results and the `4 alpha pic` folder stores the generated graphs. The current file also includes the code for plotting graphs, which might increase the difficulty in adapting this code to other data. It is recommended to separate the codes for statistical analysis and plotting.

```

1 import os
2 # ensure working directory
3 cwd = os.path.dirname(os.path.realpath(__file__))
4 os.chdir(cwd)
5 import pandas as pd
6 import numpy as np
7 import shutil
8 from scipy import stats
9 import matplotlib.pyplot as plt
10 from PIL import Image
11 import itertools
12
13 # do the analysis on the specific folder
14 dir_path = f'2 alpha/'
15 output_dir = f'3 alpha_stat/'
16 output_pic_dir = os.path.join(cwd, '4 alpha pic')
17
18 # Specify the frequency range for alpha range
19 cut_off_component = 15 # the last 15 Hz is not used
20 max_Hz = 40 - cut_off_component
21 freqs = np.array(range(2, 41-cut_off_component)) # 2 to 25 Hz
22
23
24 # Check if output_dir is not empty and it exists
25 if output_dir and os.path.exists(output_dir):
26     # Remove the directory and its contents
27     if os.path.isdir(output_dir):
28         shutil.rmtree(output_dir)
29     else:
30         os.remove(output_dir)
31 # Recreate the directory
32 os.makedirs(output_dir)
33 os.makedirs(output_pic_dir, exist_ok=True)
34
35 # Get a list of all files in the directory
36 all_files = os.listdir(dir_path)

```

Redundant peaks that do not falls within the alpha frequency band (CF <= 8.2 Hz or CF >= 13.5 Hz) were filtered out and stored in [original file name]_peak_else_sub.csv.

```

55 peak_else = linear_result[(linear_result['CF'] < 8.2) | (linear_result['CF'] > 13.5)]
56 peak_else = pd.concat([peak_else, no_peak], axis=0)
57 peak_else.to_csv(f'{output_dir}{file_name}_peak_else_sub.csv')

```

And we replaced the filtered data with CF = None, PW and BW = 0. Fittings with $r^2 \leq 0.7$ were also filtered out.

```

59 # Replace 'CF' values with NaN and 'PW' with 0
60 linear_result.loc[peak_else.index, 'CF'] = np.nan
61 linear_result.loc[peak_else.index, 'PW'] = 0
62 linear_result.loc[peak_else.index, 'BW'] = 0
63 linear_result = linear_result.drop(no_peak.index)
64 # filter out the participants with r_squared < 0.7
65 linear_result = linear_result[linear_result['r_squared'] > 0.7]

```

For the circumstances in which one participant shows multiple peaks in one condition, the peak with largest PW was kept. The filtered data is stored as [original file name]_stat_data.csv

```

66     # keep the one with largest PW
67     linear_result = linear_result.loc[linear_result.groupby(['PID', 'exp', 'load', 'age'])['PW'].idxmax()]
68     linear_result.reset_index(drop=True, inplace=True)
69     linear_result.to_csv(f'{output_dir}{file_name}_stat_data.csv')
70     # Append the data to linear_result
71     stat_data = pd.concat([stat_data, linear_result])

```

The `stat_data` is the dataframe containing the electrode data in the alpha band for all participants and conditions. The parameters were then averaged by subjects and conditions to avoid possible missing values in some channels. But if all channels are the missing values, the data here would still be `CF` = None, `PW` and `BW` = 0.

```

74     # Group the data by 'PID', 'exp', 'load', and 'age', and calculate the mean
75     mean_values = stat_data.groupby(['PID', 'exp', 'load', 'age'])[['CF', 'PW', 'BW', 'offset', 'exponent']].mean()

```

Additionally, we have written a script to verify whether the averaged dataframe includes data for each subject and condition. You may adjust the parameters in this script to align with your experimental design.

```

80     # Count the number of occurrences of each 'PID' value
81     pid_counts = mean_values['PID'].value_counts()
82     if any(pid_counts != 4):
83         print("Alert: Some PID values do not appear exactly 4 times!")
84     else:
85         print("All PID values appear exactly 4 times.")

```

And the averaged dataframe is stored as `mean_alpha_params.csv` in the output folder:

PID	exp	load	age	CF	PW	BW	offset	exponent
1104	wmc	1b	0	9.111439	0.79115	3.125708	1.884378	1.992298
1104	wmc	2b	0	8.9996	0.862575	3.457742	1.89228	1.997547
1104	wmv	1b	0	9.152957	0.779387	3.128941	1.832608	1.89081
1104	wmv	2b	0	8.956187	0.795952	3.452138	1.914524	1.92531
1108	wmc	1b	0	9.824598	0.757757	3.356074	1.391694	1.593638
1108	wmc	2b	0	9.976875	0.657001	3	1.497829	1.607481
1108	wmv	1b	0	9.949496	0.692775	3.214893	1.295852	1.555452
1108	wmv	2b	0	9.505782	0.737382	3.216818	1.414169	1.590604
1605	wmc	1b	0	10.43284	0.923922	4.523829	1.733701	1.65032
1605	wmc	2b	0	11.13728	0.879257	4.597858	1.534933	1.496475
1605	wmv	1b	0	10.63117	0.931301	5.116868	1.600791	1.594488
1605	wmv	2b	0	10.64825	0.859315	4.626529	1.577487	1.555034
1606	wmc	1b	0	9.243194	1.294268	4.492459	1.783159	1.793282
1606	wmc	2b	0	9.518963	1.271434	4.410441	1.806419	1.690295
1606	wmv	1b	0	9.694062	1.190672	4.282302	1.70836	1.642766
1606	wmv	2b	0	9.745809	1.089922	4.737176	1.777262	1.723217
1607	wmc	1b	0	9.910307	1.022835	3	1.634987	1.627483
1607	wmc	2b	0	9.925508	1.058999	3	1.437959	1.564488
1607	wmv	1b	0	9.95716	0.910821	3.020921	1.704068	1.740061
1607	wmv	2b	0	9.759823	0.939791	3	1.619358	1.638425
1608	wmc	1b	0	9.624807	0.874559	3.049059	1.648259	1.825907
1608	wmc	2b	0	9.597902	1.028012	3.062005	1.617983	1.722185
1608	wmv	1b	0	9.419834	0.931944	3.059113	1.692333	1.787471
1608	wmv	2b	0	9.364059	0.838634	3	1.60155	1.666319
1609	wmc	1b	0	10.1672	0.893192	5.229021	1.522548	1.88325
1609	wmc	2b	0	10.80185	0.720836	5.129145	1.541438	1.903625
1609	wmv	1b	0	10.10228	0.873563	5.054841	1.544866	1.942119
1609	wmv	2b	0	9.230868	0.993245	3.26251	1.610308	1.973586
1612	wmc	1b	0	9.550028	0.689377	4.217022	1.617928	1.733647
1612	wmc	2b	0	9.092246	0.898633	3.242637	1.507414	1.644059
1612	wmv	1b	0	9.276954	0.973843	3.391871	1.853235	1.963609
1612	wmv	2b	0	9.415717	0.685678	3	1.5478	1.70808
1615	wmc	1b	0	9.976919	0.639931	3.086872	1.520533	1.539616

Statistical testing

Our research interest is whether the n-back load modulated the periodic oscillations so we want to test if these parameters are significantly different between loads. To do this, we subtracted the five parameters from load 2b condition by 1b condition, then perform statistical analyses for each age group and each parameter. This data is stored as `mean_alpha_delta_nback_params.csv` and the t-test result stores in `one_sample_t_test.csv`.

plotting results

To visualize the filtered data and statistical results, we utilize the functions from Thomas's [FOOF](#) package. A function to lighten the color for each subject's curve was also included. More details for plotting are covered below.

Fig 4 alpha.png

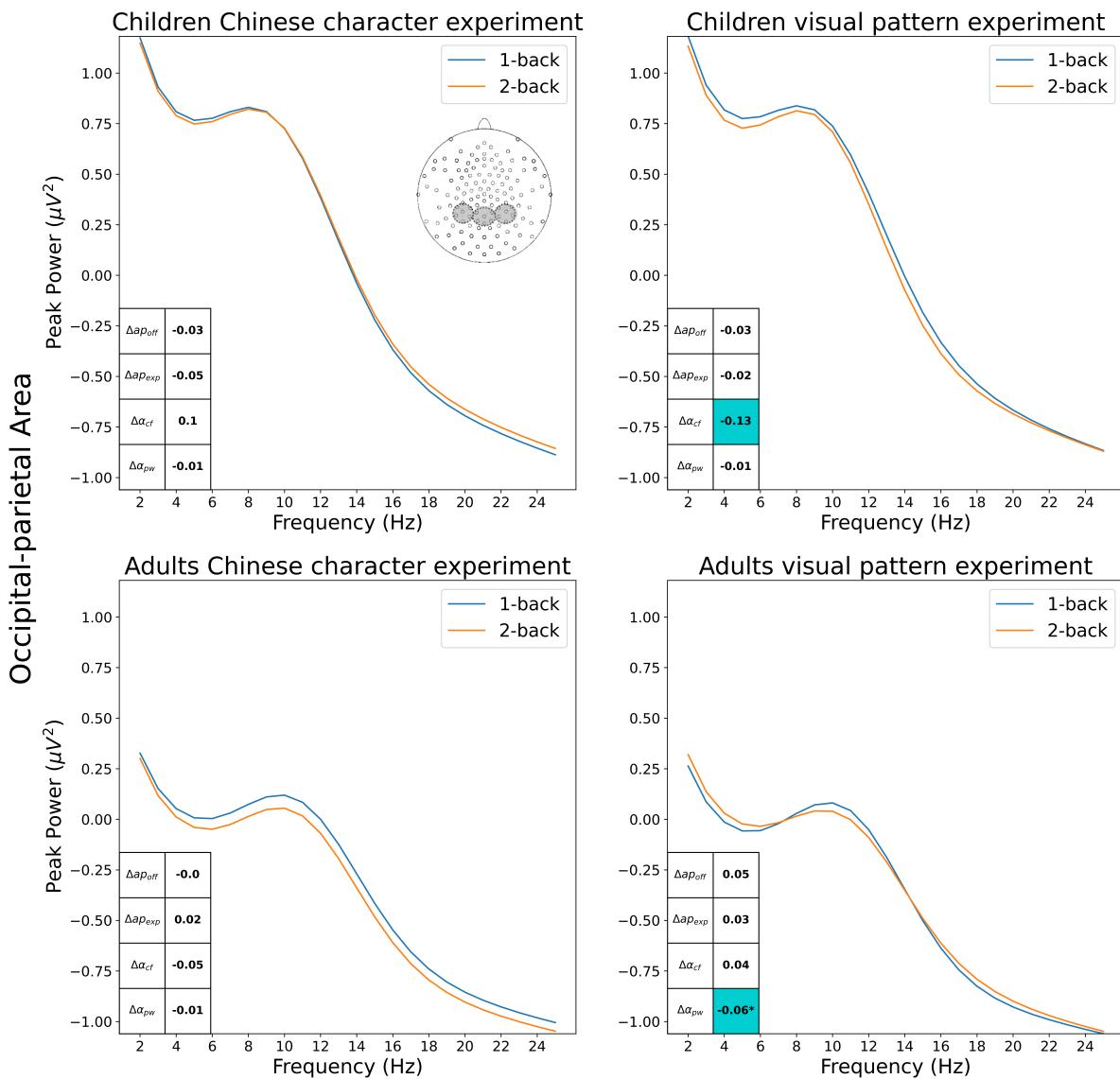


Fig 3 a.png

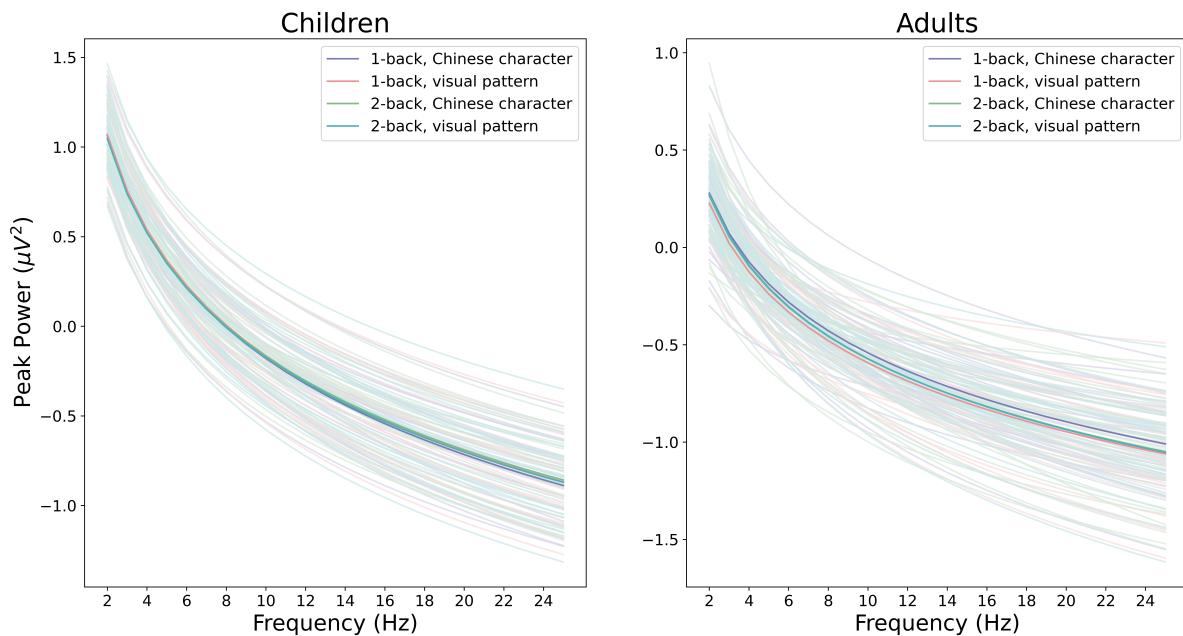
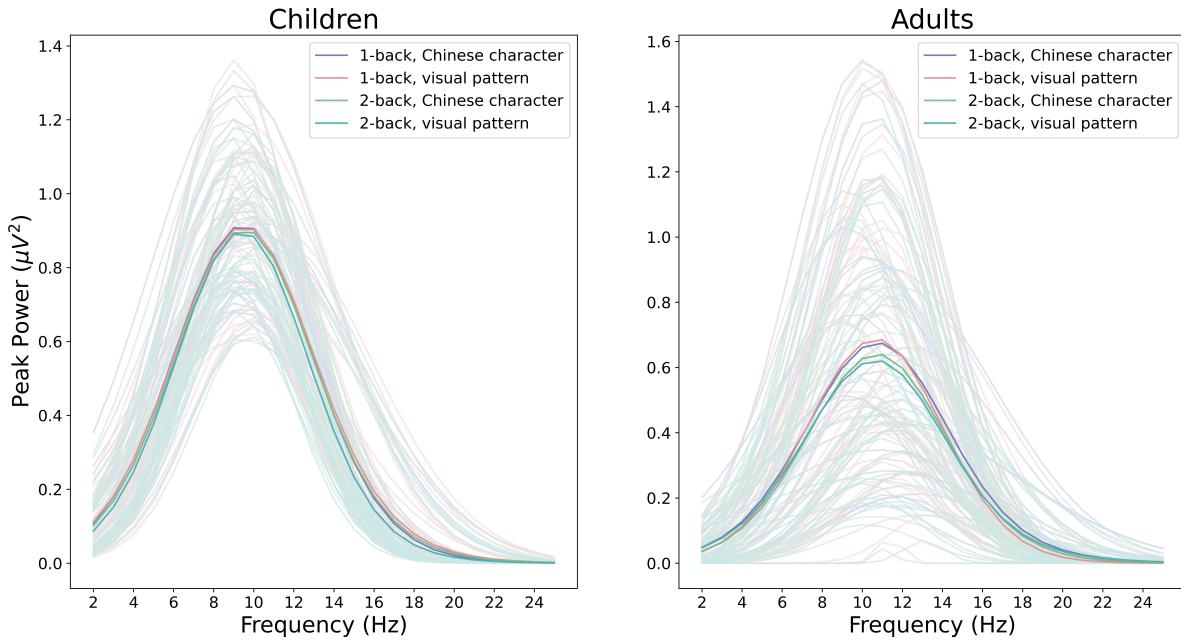


Fig 3 g.png



The functions for plotting:

```

159  # the function to calculate the aperiodic power values
160  def calculate_y_ap(group):
161      offset = np.repeat(group['offset'].values, len(freqs))
162      exp = np.repeat(group['exponent'].values, len(freqs))
163      y_ap = offset - np.log10(freqs ** exp)
164      return y_ap
165
166  # the function to calculate the periodic power values
167  def calculate_y_p(group):
168      cf = np.repeat(group['CF'].values, len(freqs))
169      pw = np.repeat(group['PW'].values, len(freqs))
170      bw = np.repeat(group['BW'].values, len(freqs))
171
172      ys = np.zeros_like(freqs)
173      y_p = ys + pw * np.exp(-(freqs - cf)**2 / (2*bw**2))
174      return y_p
175
176
177  def lighten_color(color, amount=0.5):
178      """
179          Lightens the given color by multiplying (1-luminosity) by the given amount.
180          Input can be matplotlib color string, hex string, or RGB tuple.
181
182          Examples:
183          >> lighten_color('g', 0.3)
184          >> lighten_color('#F034A3', 0.6)
185          >> lighten_color((.3,.55,.1), 0.5)
186      """
187  import matplotlib.colors as mc
188  import colorsys
189
190  try:
191      c = mc.cnames[color]
192  except:
193      c = color
194  c = colorsys.rgb_to_hls(*mc.to_rgb(c))
195  return colorsys.hls_to_rgb(c[0], 1 - amount * (1 - c[1]), c[2])

```

Fig4 alpha

The data for this plot is calculated by adding the power values of periodic (`y_p` function) and aperiodic (`y_ap` function) components together. The plot data outputs as `p+ap_alpha_load.csv`, which includes each participant's data.

```

197 groups = mean_values.groupby(['PID', 'age', 'load', 'exp'])
198 # Apply the function to each group
199 ap_power = groups.apply(calculate_y_ap).reset_index()
200 p_power = groups.apply(calculate_y_p).reset_index()
201
202 sub_info = ap_power[['PID', 'age', 'load', 'exp']]
203 # Create a new DataFrame with columns named from 'V2' to 'V{max_Hz}'
204 columns = ['V' + str(i) for i in range(2, max_Hz + 1)]
205 ap_power = pd.DataFrame(ap_power[0].to_list(), columns=columns)
206 p_power = pd.DataFrame(p_power[0].to_list(), columns=columns)
207 ap_power_all = pd.concat([sub_info, ap_power], axis=1)
208 p_power_all = pd.concat([sub_info, p_power], axis=1)
209 # print(ap_power_all.head())
210
211 ######
212 # Alpha FOOOFed oscillation plot in Fig. 4 #
213 #####
214
215 # Merge raw_data_eeg and ap_power_all on the group columns
216 merged_df = pd.merge(p_power_all, ap_power_all, on=['PID', 'age', 'load', 'exp'], suffixes=('_p', '_ap'))
217 # Subtract the ap_power_all columns from the raw_data_eeg columns
218 for i in range(2, max_Hz+1):
219     merged_df[f'V{i}_p'] = merged_df[f'V{i}_p'] + merged_df[f'V{i}_ap'] #- merged_df[f'V{i}_ap']
220
221 # Drop the original columns
222 merged_df.drop(columns=[f'V{i}_p' for i in range(2, max_Hz+1)] + [f'V{i}_ap' for i in range(2, max_Hz+1)], inplace=True)
223 merged_df.to_csv(f'{output_dir}/p+ap_alpha_load.csv', index=False)

```

Then we averaged the power values by conditions so that we have one data for and condition each frequency for visualization.

```

282 # Set the title
283 ax.set_title(f'{age_group} {experiment}', fontsize=24)
284 ax.set_xlabel('Frequency (Hz)', fontsize=20)
285 if i not in [1, 3]: # Dont plot y title for the right side of the subplot
286     ax.set_ylabel(r'Peak Power ($\mu$V2)$', fontsize=20)
287 new_ticks = np.arange(0, max_Hz-1, 2)
288 # Generate new labels for these tick marks
289 new_labels = new_ticks + 2 # starts from 2 Hz
290 # Set the new x-axis tick marks
291 ax.set_xticks(new_ticks)
292 # Set the tick labels with larger font size
293 ax.tick_params(axis='both', which='major', labelsize=14)
294 # Set the x-axis labels
295 ax.set_xticklabels(new_labels)
296 ax.legend(fontsize=18, loc='upper right')
297 # Set the same y-axis limits for all subplots
298 ax.set_ylim(global_min, global_max)
299
300 # Add an overall y-axis title
301 fig.text(0.04, 0.5, 'Occipital-parietal Area', va='center', rotation='vertical', fontsize=28)
302
303 # Get the t, p values for the plot
304 off_t = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'offset_diff'), 'T statistic'].values[0]
305 off_p = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'offset_diff'), 'P value'].values[0]
306 exp_t = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'exponent_diff'), 'T statistic'].values[0]
307 exp_p = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'exponent_diff'), 'P value'].values[0]
308 cf_t = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'CF_diff'), 'T statistic'].values[0]
309 cf_p = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'CF_diff'), 'P value'].values[0]
310 pw_t = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'PW_diff'), 'T statistic'].values[0]
311 pw_p = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'PW_diff'), 'P value'].values[0]
312 bw_t = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'BW_diff'), 'T statistic'].values[0]
313 bw_p = results_df.loc[(results_df['age'] == age) & (results_df['exp'] == exp) & (results_df['column'] == 'BW_diff'), 'P value'].values[0]
314
315 # add significance mark(*) manually
316 if i < 3:
317     cell_text = [[r'$\Delta ap_{off}$', round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'offset_diff'].mean(), 2)],
318                 [r'$\Delta ap_{exp}$', round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'exponent_diff'].mean(), 2)],
319                 [r'$\Delta \alpha_{cf}$', round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'CF_diff'].mean(), 2)],
320                 [r'$\Delta \alpha_{pw}$', round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'PW_diff'].mean(), 2)]]
321     # [r'$\Delta \alpha_{bw}$', round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'BW_diff'].values[0], 2)]]
322 else:
323     cell_text = [[r'$\Delta ap_{off}$', round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'offset_diff'].mean(), 2)],
324                 [r'$\Delta ap_{exp}$', round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'exponent_diff'].mean(), 2)],
325                 [r'$\Delta \alpha_{cf}$', round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'CF_diff'].mean(), 2)],
326                 [r'$\Delta \alpha_{pw}$', str(round(result.loc[(result['age'] == age) & (result['exp'] == exp), 'PW_diff'].mean(), 2)) + '*']]
```

What's worth noticing is that we have a box at the left bottom of each subplot. It's the t values in the one sample t test. The text in the left column is formatted using LaTeX to visually represent mathematical expressions, such as Δap_{off} . Note that the asterisks for significance were not added automatically.

Instead, cell colors were used to indicate significant results. Which can help you identify which cell to add the mark. Orange cell color indicates .05 level significance at $t > 0$, while blue indicates .05 level significance at $t < 0$. This code also add a montage picture in gray scale from the `montage` `pic/alpha.png` at the first subplot.

```
328     table = ax.table(cellText=cell_text, loc='bottom left', bbox=[0, 0, 0.2, 0.4])
329     # Set the font size of the table
330     table.set_fontsize(20)
331
332     cells = table.get_celld()
333     for cell in cells.values():
334         cell.set_text_props(horizontalalignment='center', weight='bold')
335
336     # Center get bg color for the significance
337     for key, cell in table.get_celld().items():
338         t_list = [off_t, exp_t, cf_t, pw_t, bw_t]
339         p_list = [off_p, exp_p, cf_p, pw_p, bw_p]
340         for i, (t, p) in enumerate(zip(t_list, p_list)):
341             if key[1] == 1 and key[0] == i: # key[1] is the column index, key[0] is the row index
342                 if p <= 0.05 and t > 0:
343                     cell.set_facecolor('darkorange')
344                 elif p <= 0.05 and t < 0:
345                     cell.set_facecolor('darkturquoise')
346
347     # Add the head map image to each subplot
348     first_subplot = True
349     for ax in axs.flat:
350         if first_subplot:
351             # Create a new axes for the image in the current subplot
352             img_ax = ax.inset_axes([0.62, 0.48, 0.35, 0.35])
353             img_ax.imshow(img)
354             img_ax.axis('off') # Hide the axis
355             first_subplot = False
356
357     plt.savefig(os.path.join(output_pic_dir, f'Fig4_alpha.png'), dpi=500, bbox_inches='tight')
358     # plt.show()
```

The final figures are shown below:

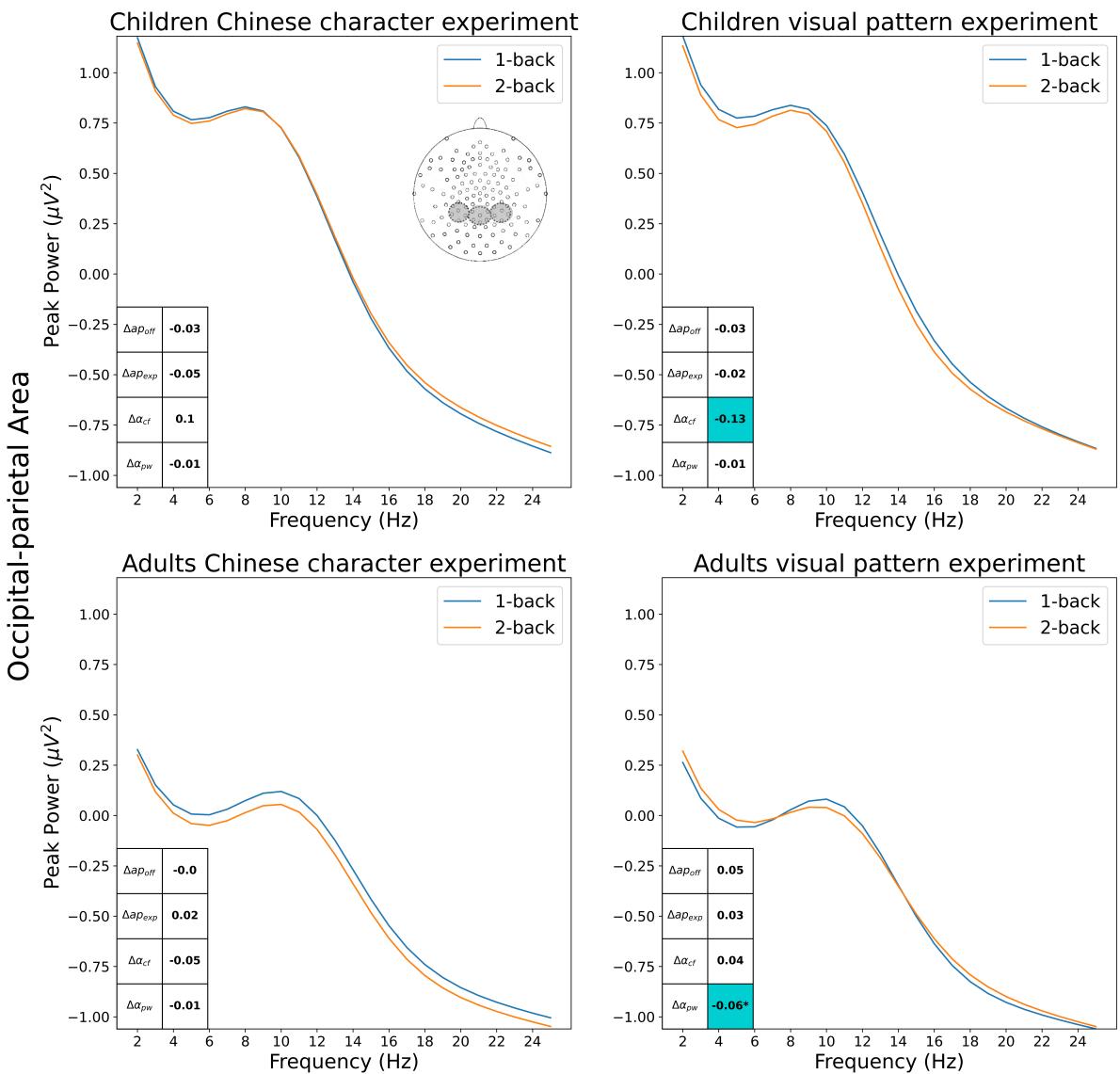


Fig 3 a & Fig 3 g

This plot is about plotting the aperiodic components in each condition each group., you can adjust the settings based on your need. The data input is the same as the previous plot, except the periodic component in the merged_df is subtracted (see row 367).

```

360 ##########
361 # plot the aperiodic power load in Fig. 3 #
362 #####
363
364 merged_df = pd.merge(p_power_all, ap_power_all, on=['PID', 'age', 'load', 'exp'], suffixes=('_p', '_ap'))
365 # Subtract the ap_power_all columns from the raw_data_eeg columns
366 for i in range(2, max_Hz+1):
367     merged_df[f'V{i}_p'] = merged_df[f'V{i}_p'] + merged_df[f'V{i}_ap'] - merged_df[f'V{i}_p']
368
369 # Drop the original columns
370 merged_df.drop(columns=[f'V{i}_p' for i in range(2, max_Hz+1)] + [f'V{i}_ap' for i in range(2, max_Hz+1)], inplace=True)
371 # print(merged_df.head())
372 merged_df.to_csv(f'{output_dir}/ap_alpha_load.csv', index=False)
373 # calculate the mean value for each column startswith V by groups_mean
374 groups_mean = merged_df.groupby(['age', 'load', 'exp'])

```

The selected color are ['#797BB7', '#E79397', '#80BA8A', '#51B1B7'], two warm color and two cold color, which were lightened by 70% for each subject's line plot.

```

388 # Loop over the conditions and plot each condition in a separate subplot
389 for i, (ax, (age)) in enumerate(zip(axes, conditions)):
390     # Select the data for this age
391     data = mean_df.xs(age, level='age')
392     # Define the colors
393     colors = ['#797BB7', '#E79397', '#80BA8A', '#51B1B7']
394     # Create a cycle of colors
395     color_cycle = itertools.cycle(colors)
396     # Lighten the colors by 70%
397     light_colors = [lighten_color(color, 0.3) for color in colors]
398     # Create a cycle of light colors
399     light_color_cycle = itertools.cycle(light_colors)
400
401     # Each subject data
402     PID_data = PID_df.xs(age, level='age')
403
404     # Group by the first two levels of the index
405     for (level0_value, level1_value), group in PID_data.groupby(level=[0, 1]):
406         # 'group' is a subset of PID_data where the first level of the index equals 'level0_value'
407         # and the second level of the index equals 'level1_value'
408         # Get the next color in the cycle
409         color = next(light_color_cycle)
410         for pid in group.index:
411             # Plot the data for this group
412             ax.plot(group.loc[(pid, slice(None))], color=color, alpha=0.7)

```

For periodic components, we subtracted the aperiodic components in the data:

```

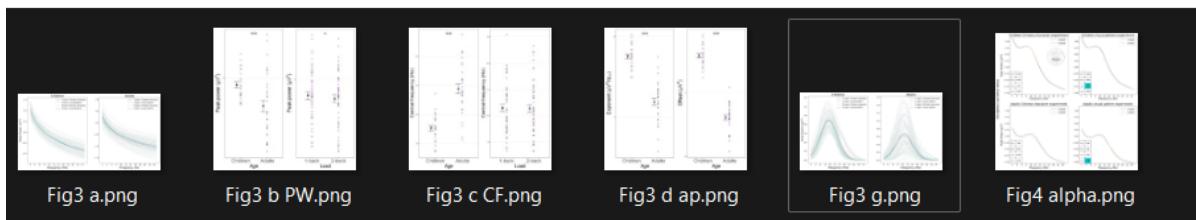
465 #####
466 # plot the periodic power load in Fig. 3 #
467 #####
468
469 merged_df = pd.merge(p_power_all, ap_power_all, on=['PID', 'age', 'load', 'exp'], suffixes=('_p', '_ap'))
470 # Subtract the ap_power_all columns from the raw_data_eeg columns
471 for i in range(2, max_Hz+1):
472     merged_df[f'V{i}_p'] = merged_df[f'V{i}_p'] + merged_df[f'V{i}_ap'] - merged_df[f'V{i}_ap']

```

The above code was not sealed in a function for easy modification. Further details and examples can be found in the R script below.

3 alpha plot

The remaining 3 graphs in the `4 alpha pic` folder were plotted in R. `ggplots2` package in R is highly recommended for statistical plots for the best visual experience.



In this code the coder used a self-wrote function again called `summarySE.R`. Ifor descriptive statistics. You can select other functions or packages according to your preference.

```

1 library(lmerTest)
2 library(rstatix)
3 library(dplyr)
4 library(bruceR)
5 library(gridExtra)
6
7
8 set.wd()
9 source('summarySE.R')
10
11 stat_data<-read.csv("3_alpha_stat/mean_alpha_params.csv",header = TRUE)
12 stat_data <- stat_data %>% filter(!is.na(CF))
13 # View(stat_data)
14 stat_data$age<-as.factor(stat_data$age)
15
16 res.aov <- anova_test(
17   data=stat_data, dv = PW, wid = PID,between = age,
18   within = c(exp, load)
19 )
20 get_anova_table(res.aov)
21 "ANOVA Table (type III tests) PW
22
23      Effect DFn DFD      F      p p<.05      ges
24 1       age    1 54 9.097 0.004      * 1.35e-01
25 2       exp    1 54 0.254 0.616           1.80e-04
26 3       load    1 54 4.288 0.043      * 1.00e-03
27 4   age:exp    1 54 0.005 0.944           3.49e-06
28 5   age:load    1 54 1.114 0.296           3.80e-04
29 6   exp:load    1 54 1.512 0.224           5.10e-04
30 7 age:exp:load    1 54 1.020 0.317           3.44e-04"

```

For data pre-processing, we just calculated the mean and [se](#) for the needed parameters in the para_data. The para_data_sub stores the variables with _sub which contains participant's data for the dot plot.

```

296 #####
297 # exp & offset
298 #####
299 CF_sub_1 <- stat_data %>% summarySE(measurevar = "exponent", groupvars=c('PID', "age"))
300 CF_sub_1 <- CF_sub_1 %>%
301   rename(y_mean = exponent_mean, y_median = exponent_median) %>%
302   mutate(P_para = "exponent")
303 PW_sub_1 <- stat_data %>% summarySE(measurevar = "offset", groupvars=c('PID', "age"))
304 PW_sub_1 <- PW_sub_1 %>%
305   rename(y_mean = offset_mean, y_median = offset_median) %>%
306   mutate(P_para = "offset")
307 sub_p <- bind_rows(CF_sub_1, PW_sub_1)
308
309 CF_1 <- summarySE(CF_sub_1, measurevar = "y_mean", groupvars=c("age"))%>%
310   mutate(P_para = "exponent")
311 PW_1 <- summarySE(PW_sub_1, measurevar = "y_mean", groupvars=c("age"))%>%
312   mutate(P_para = "offset")
313 mean_p <- bind_rows(CF_1, PW_1)
314
315 # Exponent plot
316 para_data <- mean_p %>% filter(P_para == "exponent")
317 para_data_sub <- sub_p %>% filter(P_para == "exponent")
318
319 # Modify the data to include 'age' in 'P_para'
320 para_data <- para_data %>% mutate(P_para = paste(P_para, age, sep = "_"))
321 para_data_sub <- para_data_sub %>% mutate(P_para = paste(P_para, age, sep = "_"))

```

para_data:

	age	N	y_mean_mean	y_mean_median	sd	se	ci	P_para
1	0	25	1.757497	1.750471	0.1456513	0.02913026	0.06012190	exponent
2	1	31	1.191341	1.209402	0.2503659	0.04496704	0.09183495	exponent

para_data_sub:

	PID	age	N	y_mean	y_median	sd	se	ci	P_para
1	1104	0	4	1.9514914	1.9588039	0.05213434	0.02606717	0.08295737	exponent
2	1108	0	4	1.5867939	1.5921209	0.02214808	0.01107404	0.03524254	exponent
3	1605	0	4	1.5740793	1.5747610	0.06484406	0.03242203	0.10318137	exponent
4	1606	0	4	1.7123900	1.7067560	0.06323628	0.03161814	0.10062304	exponent
5	1607	0	4	1.6426140	1.6329540	0.07267755	0.03633877	0.11564620	exponent
6	1608	0	4	1.7504706	1.7548282	0.07057193	0.03528596	0.11229568	exponent
7	1609	0	4	1.9256451	1.9228719	0.04021603	0.02010801	0.06399267	exponent
8	1612	0	4	1.7623486	1.7208633	0.13936402	0.06968201	0.22175926	exponent
9	1615	0	4	1.5082498	1.5443498	0.07667016	0.03833508	0.12199934	exponent
10	1619	0	4	1.8389415	1.8540837	0.14470486	0.07235243	0.23025773	exponent
11	1621	0	4	1.6775884	1.6581103	0.07058767	0.03529383	0.11232073	exponent
12	1623	0	4	1.4956443	1.4784689	0.07950366	0.03975183	0.12650807	exponent
13	1625	0	4	1.9033371	1.9161550	0.09548111	0.04774055	0.15193175	exponent
14	1627	0	4	1.8378967	1.8372548	0.07604051	0.03802026	0.12099742	exponent
15	1629	0	4	1.7857024	1.7833761	0.05157240	0.02578620	0.08206319	exponent
16	1632	0	4	1.7337446	1.7366784	0.02100797	0.01050398	0.03342837	exponent
17	1634	0	4	1.9202404	1.8854998	0.11539979	0.05769990	0.18362682	exponent
18	1635	0	4	2.0267927	2.0263313	0.05018810	0.02509405	0.07986047	exponent
19	1637	0	4	1.8009358	1.7826642	0.15020187	0.07510094	0.23900469	exponent
20	1640	0	4	1.7346181	1.7812216	0.12128318	0.06064159	0.19298860	exponent

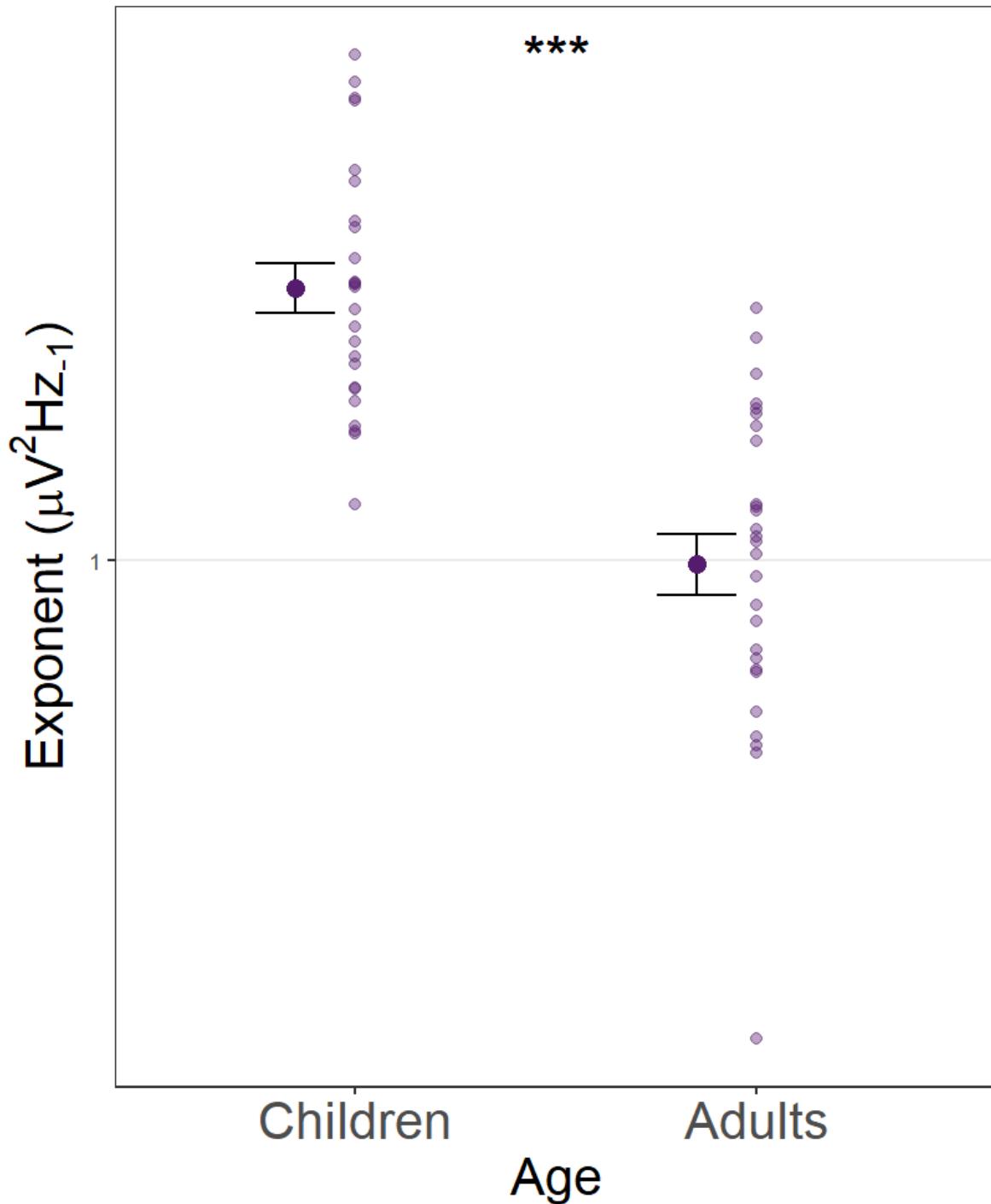
The plot contains the following parts: The error bar was 0.15 offset to left to prevent overlap. The dot at the middle of the bar indicates the mean value for each group. The jitter dots for each participants for extreme values identification; and manually added the significance asterisks (***)�.

```

319 # Modify the data to include 'age' in 'P_para'
320 para_data <- para_data %>% mutate(P_para = paste(P_para, age, sep = "_"))
321 para_data_sub <- para_data_sub %>% mutate(P_para = paste(P_para, age, sep = "_"))
322
323 p7 <- ggplot(para_data, aes(x = P_para, y = y_mean_mean)) +
324   geom_errorbar(aes(x = P_para, y = y_mean_mean,
325                      ymin = y_mean_mean-se, ymax = y_mean_mean+se),
326                      width = .2, position = position_nudge(x = -0.15)) +
327   geom_point(aes(x = P_para, y = y_mean_mean), position = position_nudge(x = -0.15),
328              shape=21, size=3, color="#551C6E", fill="#551C6E") +
329   geom_jitter(data = para_data_sub, aes(x = P_para, y = y_mean),
330                position = position_jitter(width = 0), shape=21,
331                size = 2, alpha=0.4, color="#551C6E", fill="#551C6E", show.legend=F) +
332   # labs(title = 'exponent by age') +
333   ylab(expression('Exponent '*'('mu*'V'^'2'*'Hz'['-1']*)'))+
334   scale_y_continuous(breaks = seq(0, max(para_data_sub$y_mean))) +
335   xlab('Age')+
336   # Note p significance manually
337   geom_text(data = para_data, x = 1.5, y = max(para_data_sub$y_mean),
338             label = "***", size = 8) + # p < .001
339   scale_x_discrete(labels = c("exponent_0" = "children", "exponent_1" = "Adults")) + # Add x-axis labels
340   theme_bw() + # Change theme to black and white
341   theme(
342     panel.grid.major.x = element_blank(),
343     panel.grid.minor.x = element_blank(),
344     legend.title = element_blank(),
345     legend.position = "none",
346     strip.text = element_text(size = 20), # Increase size for strip text
347     legend.text = element_text(size = 20), # Increase size for legend text
348     axis.text.x = element_text(size = 20), # Increase size for axis text
349     plot.title = element_text(size = 24, hjust = 0.5), # Increase size for plot title
350     axis.title = element_text(size = 20) # Increase size for axis title
351   )
352 p7

```

Example plot for half of Fig3 d:



Two graphs were merged together during output for compactness:

```
png("4 alpha pic/Fig3 d ap.png", width = 8, height = 8, units = "in", res = 350)
grid.arrange(p7, p8, ncol = 2)
dev.off()
```

Fig3 d:

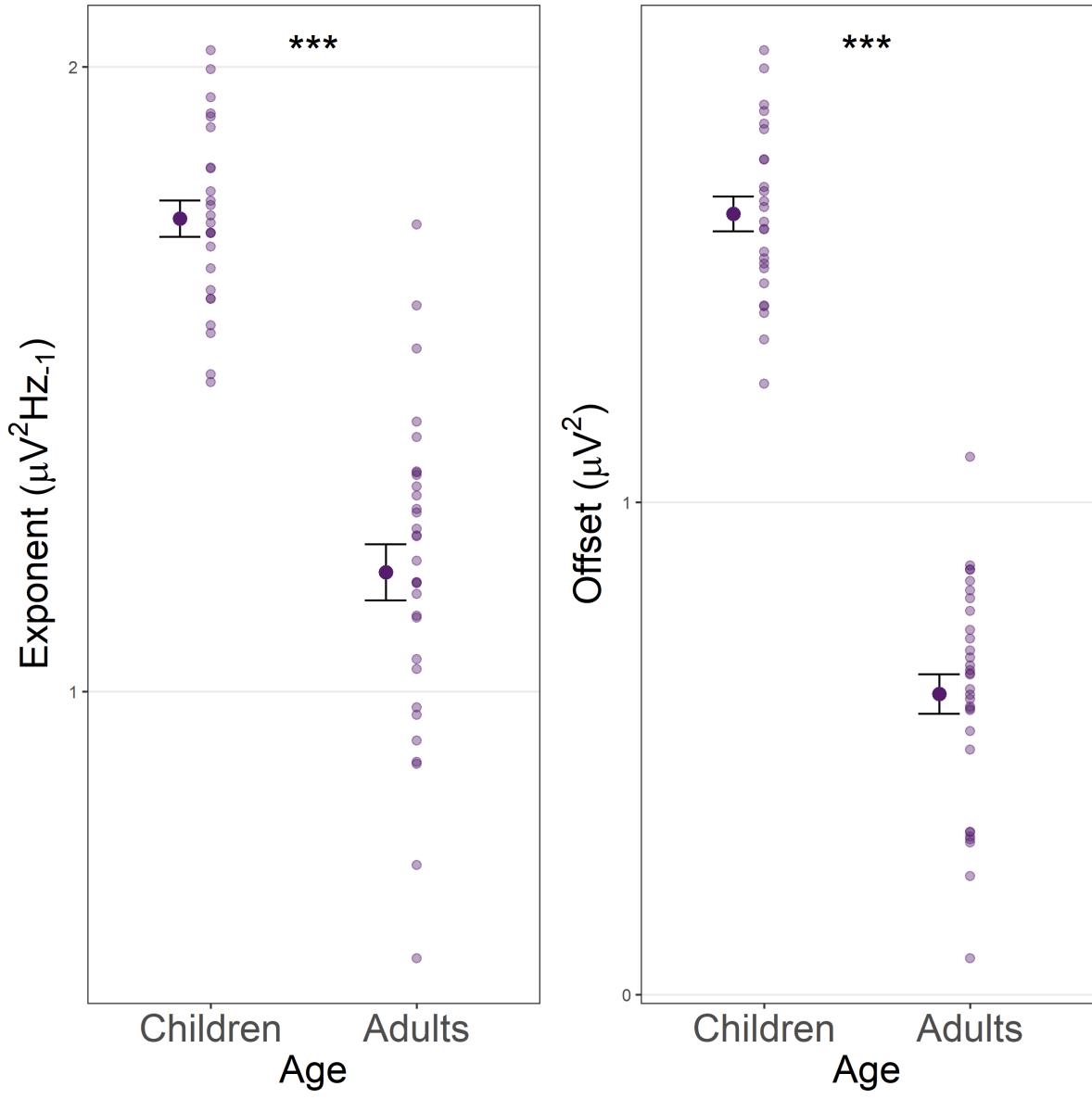
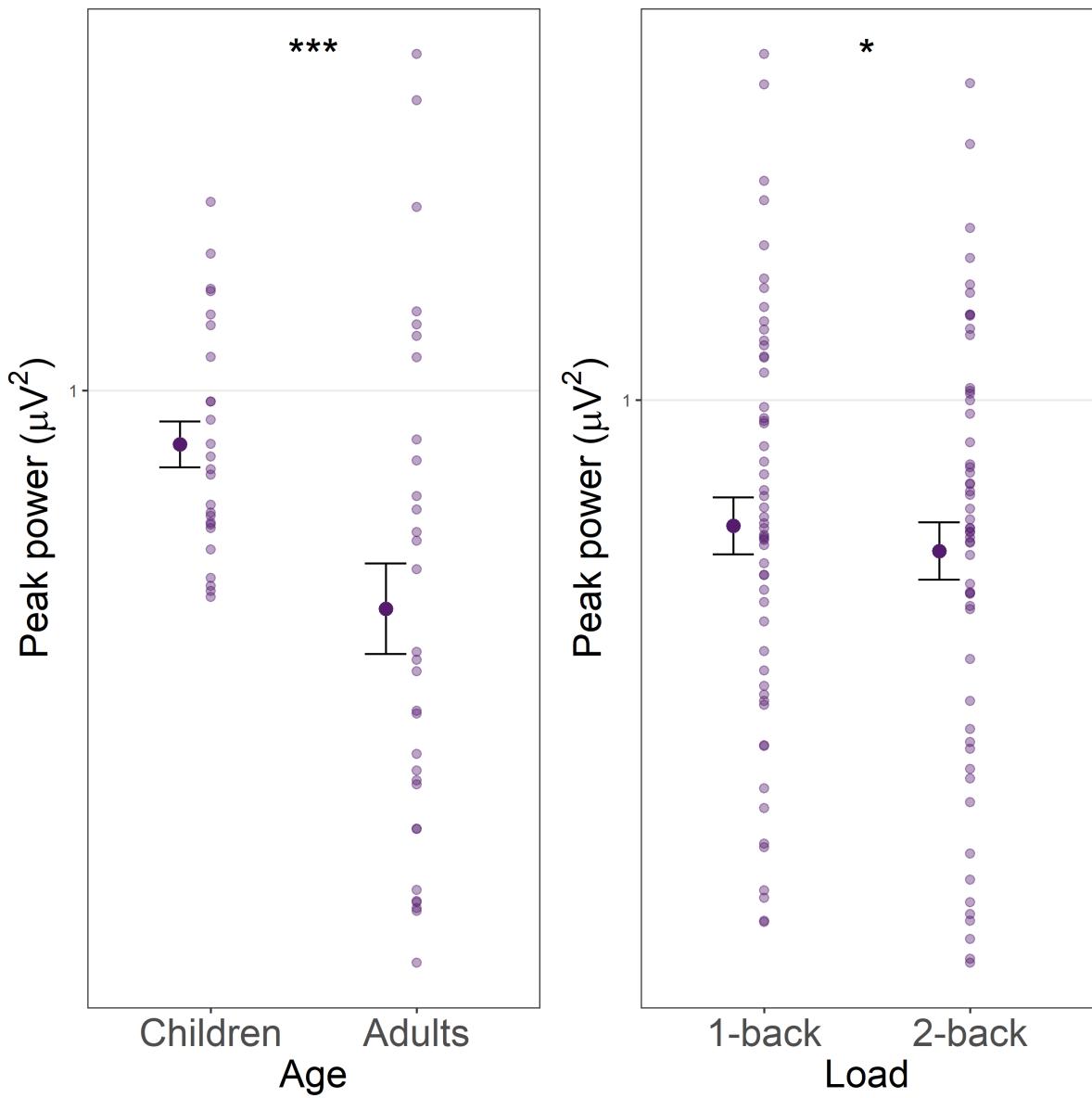
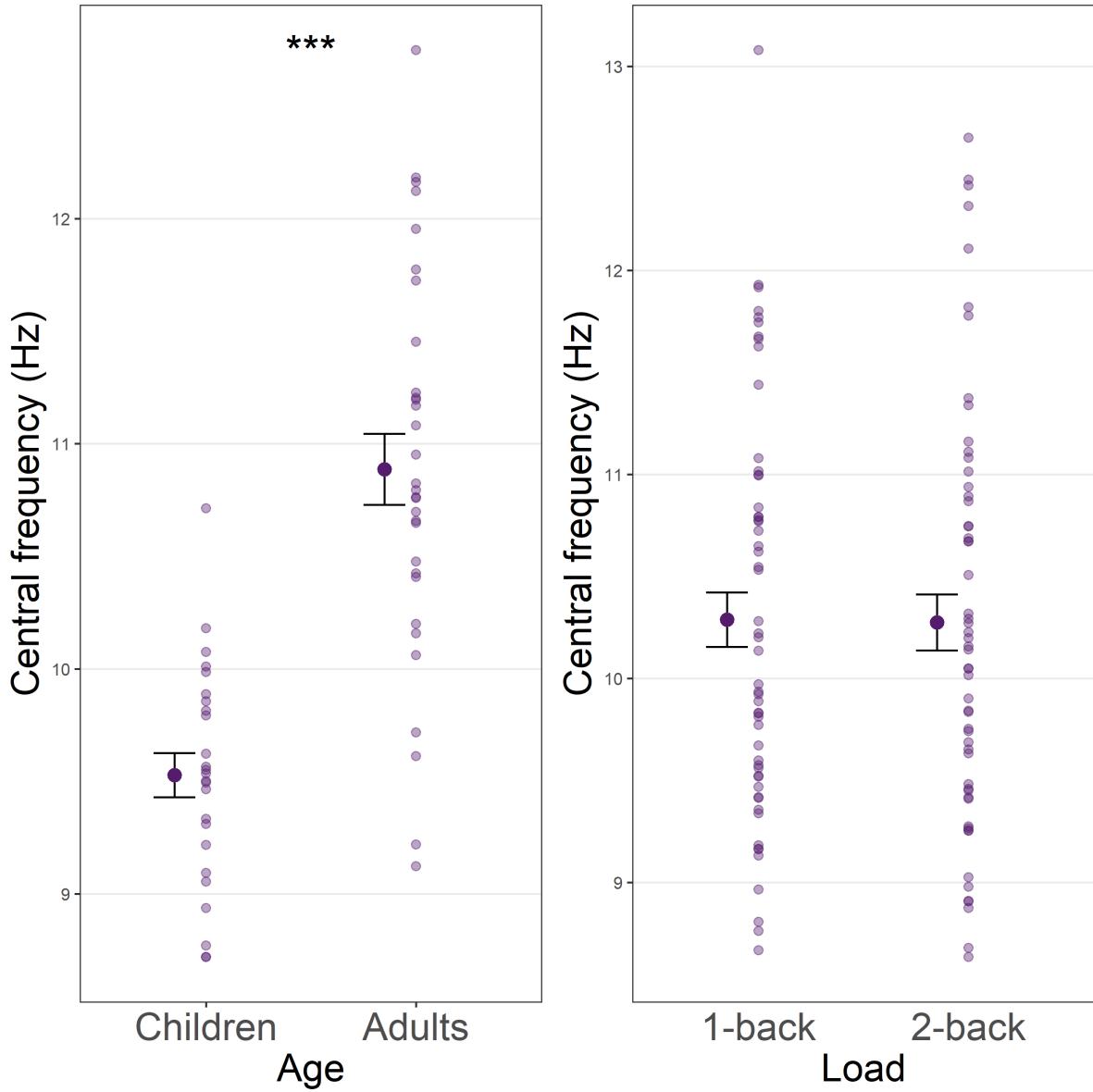


Fig3 b and Fig 3 c:





Topography data preprocess

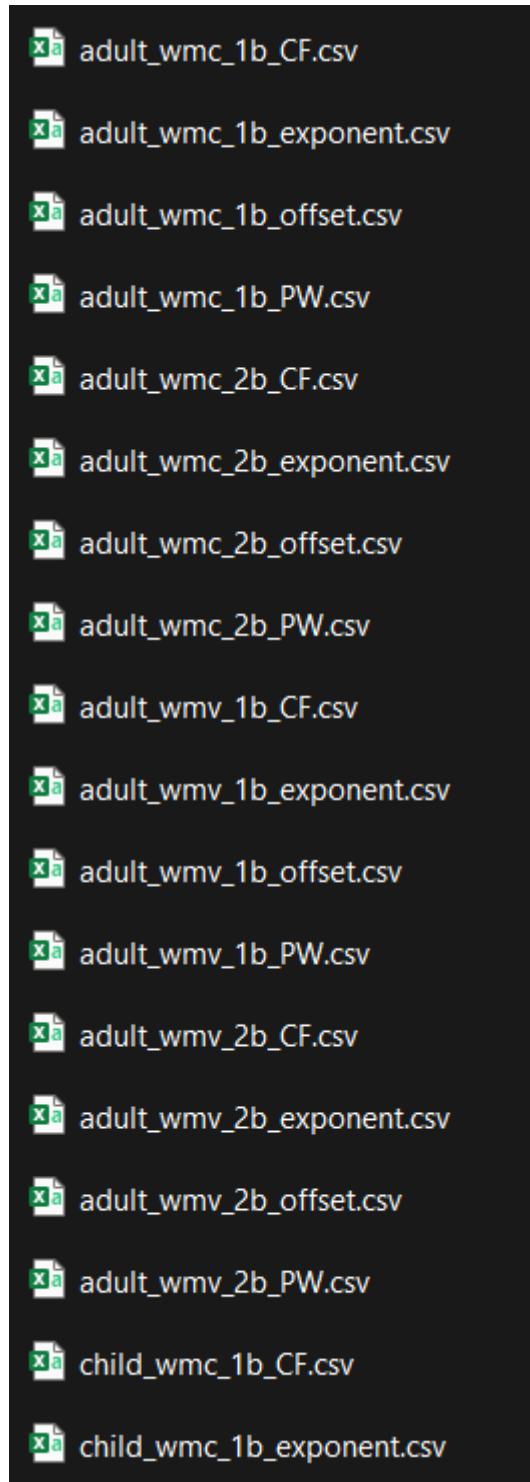
The `4 foof topo.py` and `5 topo data.py` are the two files that use all channel data from `topography data` and output `FOOOF`ed data in `5 alpha topo` and `5 theta topo`. The `5 topo cleaned` stores cleaned data for topography plots. The `topography data` folder has all channel data in it:

EEG1_all_raw.csv	EEG20_all_raw.csv	EEG39_all_raw.csv	EEG58_all_raw.csv
EEG2_all_raw.csv	EEG21_all_raw.csv	EEG40_all_raw.csv	EEG59_all_raw.csv
EEG3_all_raw.csv	EEG22_all_raw.csv	EEG41_all_raw.csv	EEG60_all_raw.csv
EEG4_all_raw.csv	EEG23_all_raw.csv	EEG42_all_raw.csv	EEG61_all_raw.csv
EEG5_all_raw.csv	EEG24_all_raw.csv	EEG43_all_raw.csv	EEG62_all_raw.csv
EEG6_all_raw.csv	EEG25_all_raw.csv	EEG44_all_raw.csv	EEG63_all_raw.csv
EEG7_all_raw.csv	EEG26_all_raw.csv	EEG45_all_raw.csv	EEG64_all_raw.csv
EEG8_all_raw.csv	EEG27_all_raw.csv	EEG46_all_raw.csv	EEG65_all_raw.csv
EEG9_all_raw.csv	EEG28_all_raw.csv	EEG47_all_raw.csv	EEG66_all_raw.csv
EEG10_all_raw.csv	EEG29_all_raw.csv	EEG48_all_raw.csv	EEG67_all_raw.csv
EEG11_all_raw.csv	EEG30_all_raw.csv	EEG49_all_raw.csv	EEG68_all_raw.csv
EEG12_all_raw.csv	EEG31_all_raw.csv	EEG50_all_raw.csv	EEG69_all_raw.csv
EEG13_all_raw.csv	EEG32_all_raw.csv	EEG51_all_raw.csv	EEG70_all_raw.csv
EEG14_all_raw.csv	EEG33_all_raw.csv	EEG52_all_raw.csv	EEG71_all_raw.csv
EEG15_all_raw.csv	EEG34_all_raw.csv	EEG53_all_raw.csv	EEG72_all_raw.csv
EEG16_all_raw.csv	EEG35_all_raw.csv	EEG54_all_raw.csv	EEG73_all_raw.csv
EEG17_all_raw.csv	EEG36_all_raw.csv	EEG55_all_raw.csv	EEG74_all_raw.csv
EEG18_all_raw.csv	EEG37_all_raw.csv	EEG56_all_raw.csv	EEG75_all_raw.csv
EEG19_all_raw.csv	EEG38_all_raw.csv	EEG57_all_raw.csv	EEG76_all_raw.csv

The `4 foof topo.py` processed data from it and the output folder `5 alpha topo` has all channel's processed data:

EEG1_all_raw_alpha_result.csv	EEG20_all_raw_alpha_result.csv	EEG39_all_raw_alpha_result.csv
EEG2_all_raw_alpha_result.csv	EEG21_all_raw_alpha_result.csv	EEG40_all_raw_alpha_result.csv
EEG3_all_raw_alpha_result.csv	EEG22_all_raw_alpha_result.csv	EEG41_all_raw_alpha_result.csv
EEG4_all_raw_alpha_result.csv	EEG23_all_raw_alpha_result.csv	EEG42_all_raw_alpha_result.csv
EEG5_all_raw_alpha_result.csv	EEG24_all_raw_alpha_result.csv	EEG43_all_raw_alpha_result.csv
EEG6_all_raw_alpha_result.csv	EEG25_all_raw_alpha_result.csv	EEG44_all_raw_alpha_result.csv
EEG7_all_raw_alpha_result.csv	EEG26_all_raw_alpha_result.csv	EEG45_all_raw_alpha_result.csv
EEG8_all_raw_alpha_result.csv	EEG27_all_raw_alpha_result.csv	EEG46_all_raw_alpha_result.csv
EEG9_all_raw_alpha_result.csv	EEG28_all_raw_alpha_result.csv	EEG47_all_raw_alpha_result.csv
EEG10_all_raw_alpha_result.csv	EEG29_all_raw_alpha_result.csv	EEG48_all_raw_alpha_result.csv
EEG11_all_raw_alpha_result.csv	EEG30_all_raw_alpha_result.csv	EEG49_all_raw_alpha_result.csv
EEG12_all_raw_alpha_result.csv	EEG31_all_raw_alpha_result.csv	EEG50_all_raw_alpha_result.csv
EEG13_all_raw_alpha_result.csv	EEG32_all_raw_alpha_result.csv	EEG51_all_raw_alpha_result.csv
EEG14_all_raw_alpha_result.csv	EEG33_all_raw_alpha_result.csv	EEG52_all_raw_alpha_result.csv
EEG15_all_raw_alpha_result.csv	EEG34_all_raw_alpha_result.csv	EEG53_all_raw_alpha_result.csv
EEG16_all_raw_alpha_result.csv	EEG35_all_raw_alpha_result.csv	EEG54_all_raw_alpha_result.csv
EEG17_all_raw_alpha_result.csv	EEG36_all_raw_alpha_result.csv	EEG55_all_raw_alpha_result.csv
EEG18_all_raw_alpha_result.csv	EEG37_all_raw_alpha_result.csv	EEG56_all_raw_alpha_result.csv
EEG19_all_raw_alpha_result.csv	EEG38_all_raw_alpha_result.csv	EEG57_all_raw_alpha_result.csv

Then the `5 topo_data.py` used similar process in `2 stat_alpha` to filter peaks. The output is also similar, but only `_PW` files are need for the topography plot.



Then use the `6 topo plot.py` to plot topography. It used montage from the `montage files` and the cleaned folder data above. And it output pictures in `6 alpha topo` and `6 theta topo` folders. Each of them contains pictures for each conditions and a joint plot for all conditions. You can uncomment the codes in the `montage_fit` functions if you want to output the normalized topography.

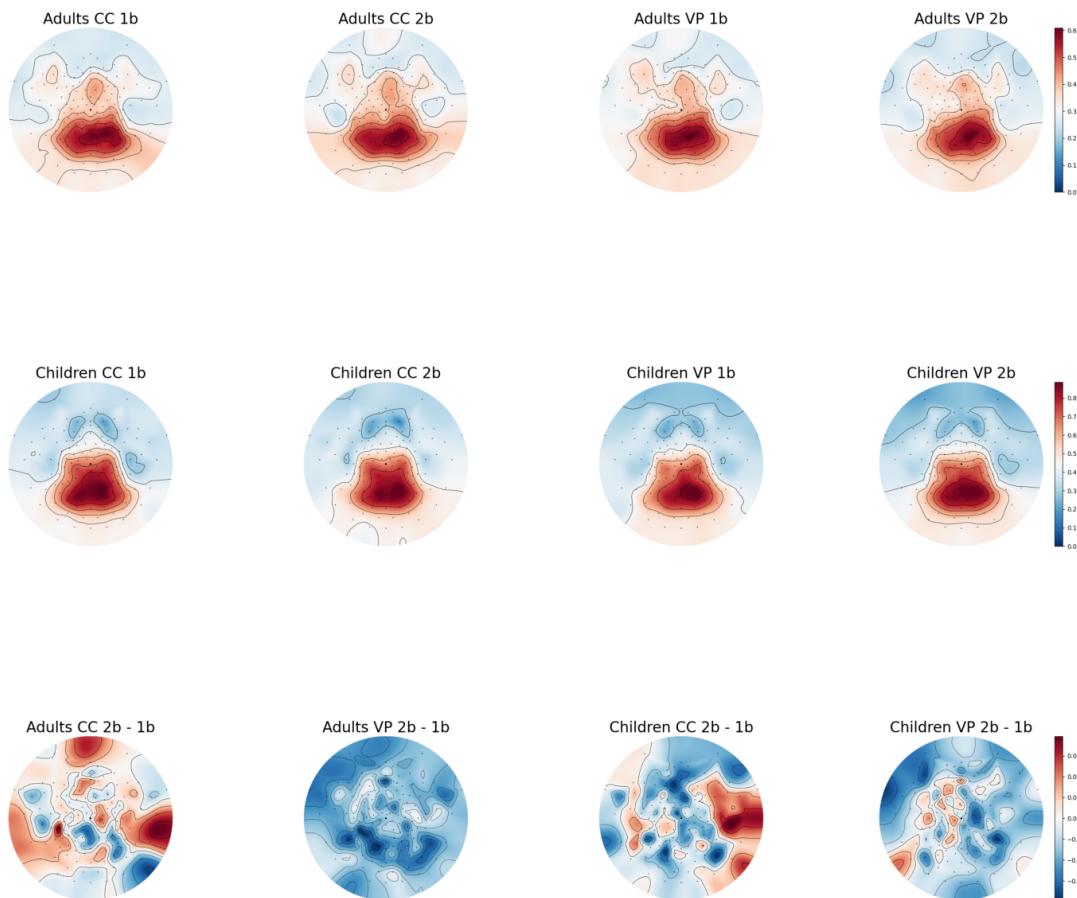
```

59 # function to fit eeg data to montage
60 def montage_fit(data):
61     # Convert the DataFrame to a 3D numpy array
62     data = data.loc[:, 'EEG1': 'EEG128'].values[:, np.newaxis]
63
64     # Initialize a new StandardScaler instance
65     scaler = StandardScaler()
66
67     # Fit the scaler to the data and transform the data for normalizing data by rows
68     # data = data.squeeze() # remove singleton dimensions
69     # data_zscore = scaler.fit_transform(data.T)
70
71     # Add an extra dimension to match the original data shape
72     # data_zscore = data_zscore.T[:, np.newaxis]

```

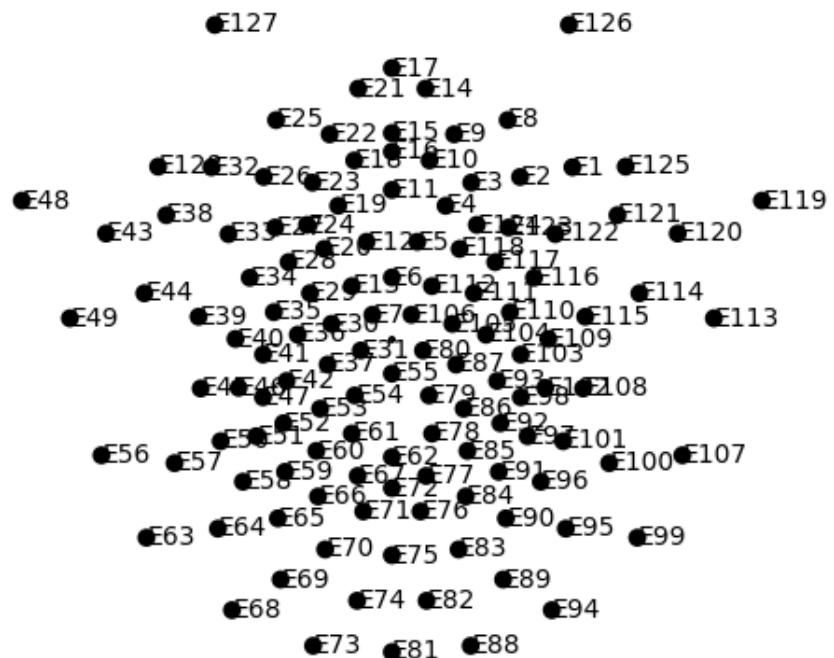
The output folder contains a joint plot for all conditions. The four plots in the same row share one color bar at the right side of each row.

Topographies for alpha band in each condition each group



It also output the sensor location plot in the `montage files`, which helps to check if the sensor location is correctly imported.

Sensor Locations with Channel Names



End & contact info

This is the end of the tutorial. The code for theta band was not included as the structure is the same. Thank you for spending so much time reading this. If you have any questions related to coding, feel free to contact the coder by quentinlampsy@gmail.com.