Python 脑电数据处理中文手册(第二版)

路子童 李婉如 聂露 赵匡是 编

编者介绍

路子童：

- 东北末流 985 软件工程专业本科，上海末流 985 认知神经科学专业硕士，俄亥俄州立大学认知神经科学专业博士在读
- 使用眼动、EEG、fMRI 和计算方法研究人脑视觉，搞技术与方法只是副业，更关心科学问题
- 不知名公众号路同学的不知名运营者，不知名工具包 NeuroRA、不知名模型 EEG2EEG 和 ReAlnet 的不知名作者
- 日常焦虑型选手，具有深夜不睡觉、沉迷追剧等宝贵缺点
- 邮箱：zitonglu1996@gmail.com

李婉如：

- 华东师范大学心理学学士，北京大学在读博士生
- 大二开始接触 EEG，给很多脑袋打过膏，洗头手法一流
- 曾经沉迷动物森友会，岛上出过大头菜 600 的高价，最近沉迷吃豆人，欢迎交友
- 不断学习中，希望能和大家讨论交流，邮箱：wanruli@stu.pku.edu.cn

聂露：

- 中山大学认知神经心理学待毕业博士一枚
- Memory and Emotion Lab 气氛组常驻嘉宾
- 致力于在情感工作记忆领域捉鬼，有必要也可以换一只
- 现在会 EEG，NIRS 和 MEG，不会的也可以学
- 希望有机会和大家请教学习，邮箱：lunie0408@gmail.com

赵匡是：

- 华东师范大学认知神经科学专业硕士，现就职博睿康
- 出生于日本，信仰起源于美国，爱好印度文化，现居中国上海，热衷台湾省资讯
- “程序员界最懂神经科学的，神经科学界最会码代码的”
- www.zhaokuangshi.cn

第二版序

三年前，我们在硕士毕业/进入工作/博士生涯开始前的暑假怀着满腔热情与忐忑发布了《Python 脑电数据处理中文手册》第一版。自那以来，我们收到了广大读者的强烈反响与宝贵意见。我们的 GitHub 项目已经收获了 232 颗 Stars，我的邮箱也时不时会有咨询相关问题的邮件（有部分邮件可能因为邮件太多忘记回复、也可能由于有些问题实在过于基础可以通过其他途径找到答案而没有回复）。

当然，第一版并不是完美的，之后会进行更新也是我们在三年前发布第一版时就有所计划的。我们希望可以不断地完善这个手册、修改旧版本里可能存在的问题、去优化与丰富我们的内容。同时，相关的依赖 Python 包也在不断更新，随之而来在手册里我们也必然需要进行一些修改。因此，这些因素都是我们更新第二版的基础。同时，非常高兴看到大家开始参考与使用这个手册，这也是对我们整个作者团队莫大的肯定。也正是大家这些正向的反馈激励我们继续前行，而带给了我们更新第二版的动力。

在第二版中，我们对手册的结构进行了重新的划分，分成了四个大的章节：单被试预处理、基础 Python 数据处理、多被试分析、以及高级脑电数据分析。我们主要修改了前一版中出现的一些笔误或描述性错误，在多被试分析章节中介绍了事件相关电位分析与时频分析，而将基于分类的脑电解码和表征相似性分析的部分归类到了高级脑电数据分析章节中，并额外增加了反向编码模型的部分。

于此同时，在三月初也发布了英文版的 Python EEG Handbook （英文版预印本见 PsyArXiv：https://osf.io/preprints/psyarxiv/dcmke），希望这一手册不仅仅在中文社区、也在英文社区帮助脑电研究人员或希望踏入脑电领域的初学者。

我们希望《Python 脑电数据处理手册》（无论中文或英文版）能一直作为一个桥梁，不仅仅连接过去和未来，也连接初学者与专家、研究者与开发者，来共同推进脑电数据分析的发展。更重要的是，我们希望通过这一本手册不仅仅能帮助你在技术上取得进步，更能激发你对认知神经科学更深的热爱与探索！

再次感谢所有人的支持，期待与你们在学术的路上相遇！

路子童 主笔

2024 年 3 月 14 日

第一版序

几乎所有脑电初学者都是从 EEGLAB 开始接触脑电预处理过程的，EEGLAB 浅显直观的 GUI 界面再或是基于 MATLAB 的代码操作影响了一代脑电人。然而，随着简洁、易上手的 Python 语言的快速发展，其丰富的社区资源也扩张到了认知神经科学领域。MNE-Python、Nilearn、Nibabel 等等相关工具包层出不穷，让我们有机会开始使用 Python 来对进行各种神经数据进行分析处理。遗憾的是，它们未能在国内迅速地普及，相对门庭冷淡。

为了弥补这一空白，一方面希望更多人加入到使用 Python 的行列中来，另一方面希望为更高阶的脑电数据操作架起桥梁，我们尝试完成一个中文版的 Python 脑电处理教程，它便是这个《Python 脑电数据处理中文手册》。

“自己编程不好怎么办？“感觉用 Python 做数据处理好难学不会怎么办？”。诚然，完全基于代码的数据处理往往会带给很多人一些担心。但是，我们希望能在这里通过我们的手册告诉你们，不需要惧怕，只要一步一步跟着学习与理解，你的编程能力一定会有所提升、你一定可以学会！

十分开心与激动，有愿意一起参与到这个手册编撰的小伙伴，很感激自己硕士三年与一群极其优秀的人在同一个 Lab 共事，也很荣幸大家与我一样愿意加入到这个自愿分享、支持开源的行动中来。一开始我们就本着严谨、真诚、公益的态度来规划这件事情，并花费了许多精力将它尽可能地做到最好，尽可能将这个手册已最好、最全面的方式呈现到各位面前。

目前，这个中文手册主要分成了两个部分，单被试预处理篇和多被试分析篇。在单被试预处理篇中，我们参考了贾会宾老师的《EEGLAB 中文手册》以及 Steven Luck 的《事件相关电位基础》，我们旨在写一个基于 Python（主要是基于 MNE-Python 工具包）的对单个被试脑电数据进行预处理的标准化流程。虽然我们努力想呈现出一个“傻瓜式”的教程，但是我们依然不建议完全没有任何编程基础和脑电基础知识的人员盲目使用。

在多被试分析篇中，我们首先从常见的 Python 数据操作、数据的读取与存储以及基础统计分析与实现等方面着手，让大家具备一定的基本数据分析能力后再进入到更高阶的数据处理中。我们也必须承认，高阶的分析在理解难度和实现难度上一定是要高于预处理部分的。但是，只要你仔细阅读每一句手册里的说明、结合注释理解每一行代码、自己亲手一步一步实现一遍，你肯定能收获很多。

衷心希望我们的手册能提供一些思路与建议，它当然无法完完全全直接适用于你自己的数据，但我们相信，只需要一些并不困难的修改之后，兴许你就能自如地开始着手处理自己的数据了。所谓“授人以鱼不如授人以渔”，望读者们能举一反三。手册里也许很多地方的内容未必深入，因此，这其中的内涵与奥秘、技巧与思想，都需要读者在不断的体验中去用心体会、在实战中不断积累经验。

完成这个中文手册花费了我们巨大的心血，在一定程度上来说他不仅仅是一个脑电的数据处理手册，更是对广大初入心理学与神经科学领域的研究人员的一个 Python 入门手册。当然，我们也希望这个中文手册是国内 Python 脑电数据处理中文手册的第一步，但也不仅仅是第一步！

十分感激读者在使用过程中给我们反馈建议与意见，也欢迎大家多多分享我们的手册让它能帮到更多人，愿此手册与我们、与读者共同进步！

路子童 主笔

2021 年 7 月 20 日

第一章： 单被试数据预处理

在此单被试分析篇中，按照脑电预处理流程，分为以下8个步骤：

- **第1步: 数据读取**
- **第2步: 滤波**
- **第3步: 去伪迹**
- **第4步: 重参考**
- **第5步: 数据分段**
- **第6步: 叠加平均**
- **第7步: 时频分析**
- **第8步: 数据提取**

第一步 - 数据读取

导入原始数据

由于大多数小伙伴熟悉的脑电数据处理工具包为基于MATLAB的EEGLAB
这里使用的数据为MATLAB经典脑电数据处理工具包EEGLAB中的经典数据"eeglab_data.set"

In [2]:
```
!pip install mne

import numpy as np
import mne
import os
import gdown
import zipfile
from mne.preprocessing import ICA
from mne.time_frequency import tfr_morlet

import warnings
warnings.filterwarnings('ignore', category=UserWarning, module='matplotlib')

%matplotlib inline

data_dir = "data/"
if not os.path.exists(data_dir):
  os.makedirs(data_dir)

# 从Google Drive下载
url = "https://drive.google.com/file/d/1bXD-_dDnH5Mv3DQrV7V9fYM4-xYsZ0DN/view?usp=sharing"
filename = "sample_data"
filepath = data_dir + filename + ".zip"

# 下载数据
gdown.download(url=url, output=filepath, quiet=False, fuzzy=True)
print("Download completes!")
# 数据解压
with zipfile.ZipFile(filepath, 'r') as zip:
    zip.extractall(data_dir)
print("Unzip completes!")

# 也可以通过百度网盘下载
# 链接:https://pan.baidu.com/s/1nQgoxeWoelDyIPduFES7YA  密码:fn96
# 下载后解压，并移动到data文件夹下

data_path = data_dir + 'sample_data/eeglab_data.set'

# 也可以直接使用你自己EEGLAB文件夹内的sample data文件夹下的数据
# data_path = "/Users/zitonglu/Desktop/EEG/eeglab14_1_2b/sample_data/eeglab_data.set"

# MNE-Python中对多种格式的脑电数据都进行了支持:
# *** 如数据后缀为.set (来自EEGLAB的数据)
#     使用mne.io.read_raw_eeglab()
# *** 如数据后缀为.vhdr (BrainVision系统)
#     使用mne.io.read_raw_brainvision()
# *** 如数据后缀为.edf
#     使用mne.io.read_raw_edf()
```

```
# *** 如数据后缀为.bdf (BioSemi放大器)
#     使用mne.io.read_raw_bdf()
# *** 如数据后缀为.gdf
#     使用mne.io.read_raw_gdf()
# *** 如数据后缀为.cnt (Neuroscan系统)
#     使用mne.io.read_raw_cnt()
# *** 如数据后缀为.egi或.mff
#     使用mne.io.read_raw_egi()
# *** 如数据后缀为.data
#     使用mne.io.read_raw_nicolet()
# *** 如数据后缀为.nxe (Nexstim eXimia系统)
#     使用mne.io.read_raw_eximia()
# *** 如数据后缀为.lay或.dat (Persyst系统)
#     使用mne.io.read_raw_persyst()
# *** 如数据后缀为.eeg (Nihon Kohden系统)
#     使用mne.io.read_raw_nihon()

# 读取数据
raw = mne.io.read_raw_eeglab(data_path, preload=True)
```

```
Requirement already satisfied: mne in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (1.6.1)
Requirement already satisfied: numpy>=1.21.2 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from mne) (1.23.5)
Requirement already satisfied: scipy>=1.7.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from mne) (1.11.1)
Requirement already satisfied: matplotlib>=3.5.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from mne) (3.6.3)
Requirement already satisfied: tqdm in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
mne) (4.65.0)
Requirement already satisfied: pooch>=1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne) (1.8.1)
Requirement already satisfied: decorator in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne) (5.1.1)
Requirement already satisfied: packaging in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne) (23.1)
Requirement already satisfied: jinja2 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (fro
m mne) (3.1.2)
Requirement already satisfied: lazy-loader>=0.3 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from mne) (0.3)
Requirement already satisfied: contourpy>=1.0.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from matplotlib>=3.5.0->mne) (1.0.5)
Requirement already satisfied: cycler>=0.10 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from matplotlib>=3.5.0->mne) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from matplotlib>=3.5.0->mne) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from matplotlib>=3.5.0->mne) (1.4.4)
Requirement already satisfied: pillow>=6.2.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from matplotlib>=3.5.0->mne) (9.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from matplotlib>=3.5.0->mne) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /Users/zitonglu/anaconda3/lib/python3.11/site
-packages (from matplotlib>=3.5.0->mne) (2.8.2)
Requirement already satisfied: platformdirs>=2.5.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-
packages (from pooch>=1.5->mne) (3.10.0)
Requirement already satisfied: requests>=2.19.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from pooch>=1.5->mne) (2.31.0)
Requirement already satisfied: MarkupSafe>=2.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pack
ages (from jinja2->mne) (2.1.1)
Requirement already satisfied: six>=1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (f
rom python-dateutil>=2.7->matplotlib>=3.5.0->mne) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/zitonglu/anaconda3/lib/python3.11/
site-packages (from requests>=2.19.0->pooch>=1.5->mne) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from requests>=2.19.0->pooch>=1.5->mne) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests>=2.19.0->pooch>=1.5->mne) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests>=2.19.0->pooch>=1.5->mne) (2023.7.22)
Downloading...
From: https://drive.google.com/uc?id=1bXD-_dDnH5Mv3DQrV7V9fYM4-xYsZ0DN
To: /Users/zitonglu/Downloads/Python-EEG-Handbook-master/data/sample_data.zip
100%|████████████████████████████████████| 7.30M/7.30M [00:00<00:00, 8.09MB/s]
Download completes!
Unzip completes!
Reading /Users/zitonglu/Downloads/Python-EEG-Handbook-master/data/sample_data/eeglab_data.fdt
Reading 0 ... 30503  =      0.000 ...   238.305 secs...
```

查看原始数据信息

In [3]:
```
print(raw)
print(raw.info)
```

```
<RawEEGLAB | eeglab_data.fdt, 32 x 30504 (238.3 s), ~7.5 MB, data loaded>
<Info | 7 non-empty values
 bads: []
 ch_names: EEG 000, EEG 001, EEG 002, EEG 003, EEG 004, EEG 005, EEG 006, ...
 chs: 32 EEG
 custom_ref_applied: False
 highpass: 0.0 Hz
 lowpass: 64.0 Hz
 meas_date: unspecified
 nchan: 32
 projs: []
 sfreq: 128.0 Hz
>
```

可以看到该脑电数据为32个导联，30504个时间点（采样率为128Hz，对应238.3s的长度）
由于读取数据时preload设为True，数据读入内存中，数据大小约为7.5MB
导联名依次为“EEG 000”, “EEG 001”, “EEG 002”, “EEG 003”等等
高通滤波0.0Hz，低通滤波64.0Hz

电极定位

这里由于导联名称不是标准的名称 如果碰到类似问题，需要手动导入脑电数据的电极位置信息 在这个例子中，即需要把数据集相关的.locs文件中的信息传入这里读取的脑电数据中

In [4]:
```
# .locs文件地址
locs_info_path = data_dir + "sample_data/eeglab_chan32.locs"
# 读取电极位置信息
montage = mne.channels.read_custom_montage(locs_info_path)
# 读取正确的导联名称
new_chan_names = np.loadtxt(locs_info_path, dtype=str, usecols=3)
# 读取旧的导联名称
old_chan_names = raw.info["ch_names"]
# 创建字典，匹配新旧导联名称
chan_names_dict = {old_chan_names[i]:new_chan_names[i] for i in range(32)}
# 更新数据中的导联名称
raw.rename_channels(chan_names_dict)
# 传入数据的电极位置信息
raw.set_montage(montage)
```

Out[4]:

▼ General

Measurement date	Unknown
Experimenter	Unknown
Participant	Unknown

▼ Channels

Digitized points	35 points
Good channels	32 EEG
Bad channels	None
EOG channels	Not available
ECG channels	Not available

▼ Data

Sampling frequency	128.00 Hz
Highpass	0.00 Hz
Lowpass	64.00 Hz
Filenames	eeglab_data.fdt
Duration	00:03:59 (HH:MM:SS)

当你的脑电电极位点为一些特定系统时，可以直接用mne.channels.make_standard_montage函数生成
以标准的国际10-20系统为例，对应代码即可改为：

> montage = mne.channels.make_standard_montage("standard_1020")

MNE中现成的其他定位系统的montage可以通过以下网址查询：
https://mne.tools/stable/auto_tutorials/intro/40_sensor_locations.html#sphx-glr-auto-tutorials-intro-40-sensor-locations-py

设置导联类型

In [5]:
```
# MNE中一般默认将所有导联类型设成“eeg”
# 这里将两个“EOG”导联的类型设定为“eog”

chan_types_dict = {"EOG1":"eog", "EOG2":"eog"}
raw.set_channel_types(chan_types_dict)
```

Out[5]:

▼ General

Measurement date	Unknown
Experimenter	Unknown
Participant	Unknown

▼ Channels

Digitized points	35 points
Good channels	30 EEG, 2 EOG
Bad channels	None
EOG channels	EOG1, EOG2
ECG channels	Not available

▼ Data

Sampling frequency	128.00 Hz
Highpass	0.00 Hz
Lowpass	64.00 Hz
Filenames	eeglab_data.fdt
Duration	00:03:59 (HH:MM:SS)

查看修改后的数据信息

In [6]:
```
# 打印修改后的数据信息

print(raw.info)
```

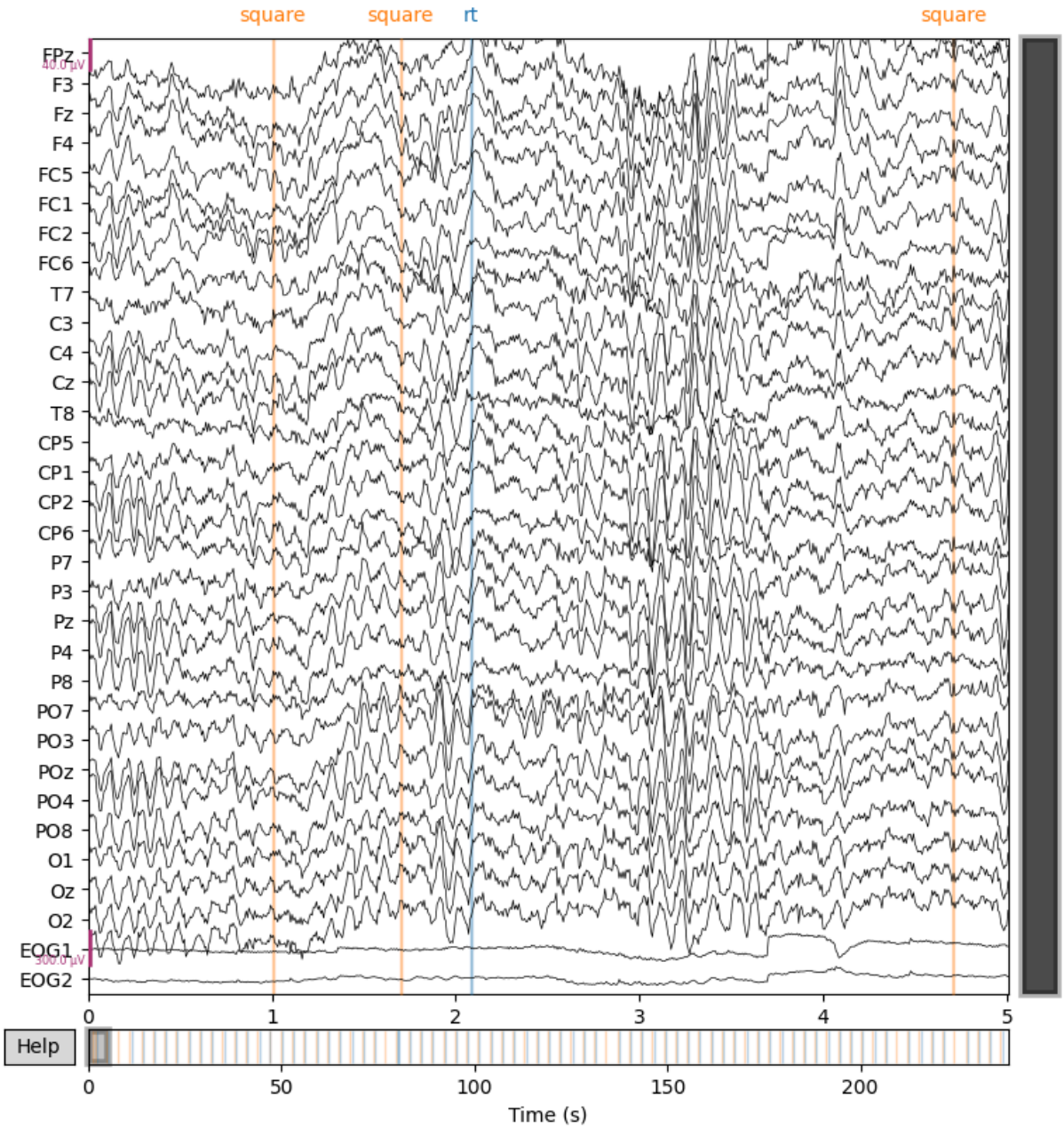
```
<Info | 8 non-empty values
 bads: []
 ch_names: FPz, EOG1, F3, Fz, F4, EOG2, FC5, FC1, FC2, FC6, T7, C3, C4, Cz, ...
 chs: 30 EEG, 2 EOG
 custom_ref_applied: False
 dig: 35 items (3 Cardinal, 32 EEG)
 highpass: 0.0 Hz
 lowpass: 64.0 Hz
 meas_date: unspecified
 nchan: 32
 projs: []
 sfreq: 128.0 Hz
>
```

可视化原始数据

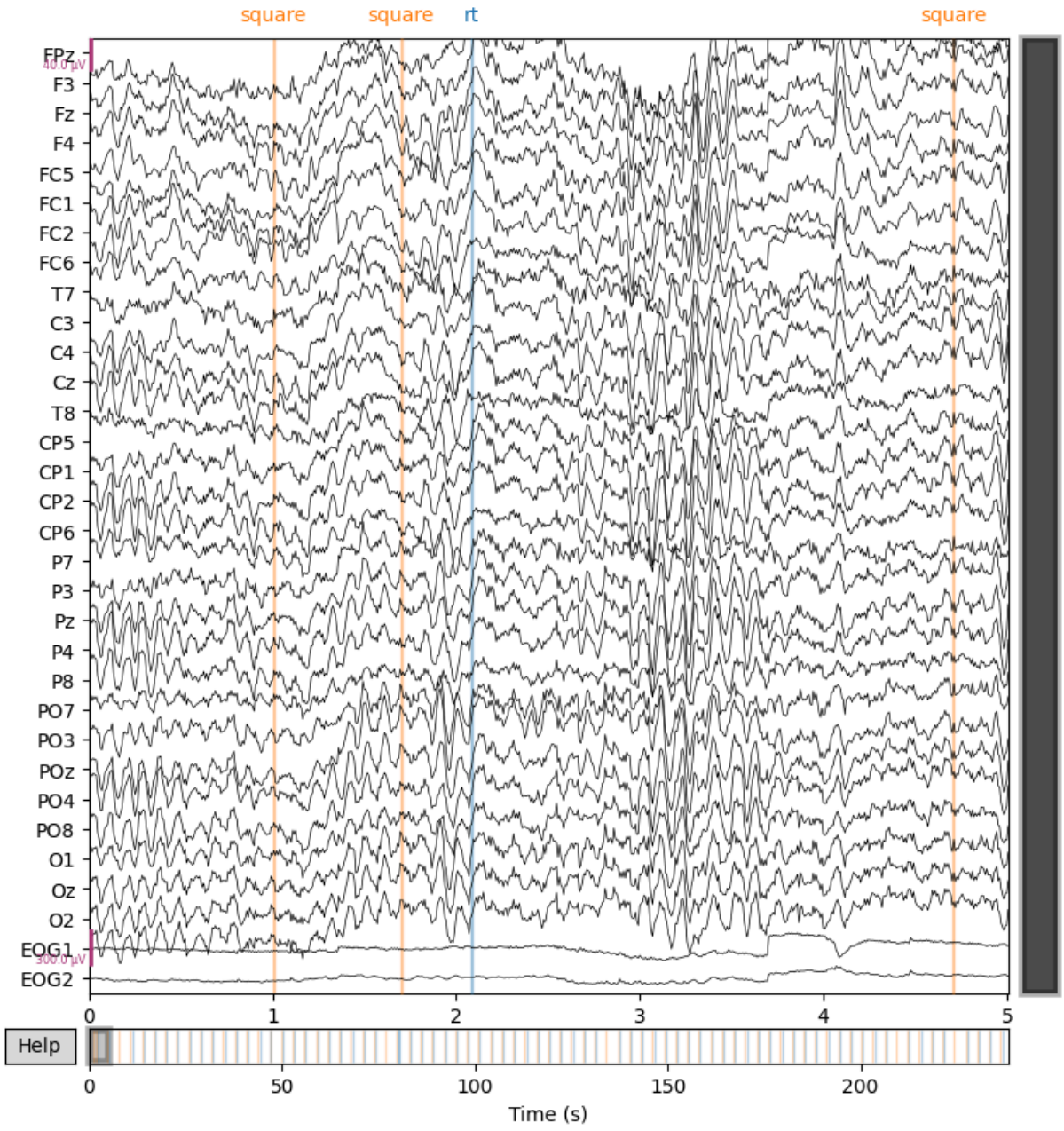
绘制原始数据波形图

In [7]:
```
raw.plot(duration=5, n_channels=32, clipping=None)
```

```
Using matplotlib as 2D backend.
```


square
square
rt
square
FPz
F3
Fz
F4
FC5
FC1
FC2
FC6
T7
C3
C4
Cz
T8
CP5
CP1
CP2
CP6
P7
P3
Pz
P4
P8
PO7
PO3
POz
PO4
PO8
O1
Oz
O2
EOG1
EOG2
40.0 µV
300.0 µV
0
1
2
3
4
5
Help
0
50
100
150
200
Time (s)

Out[7]:

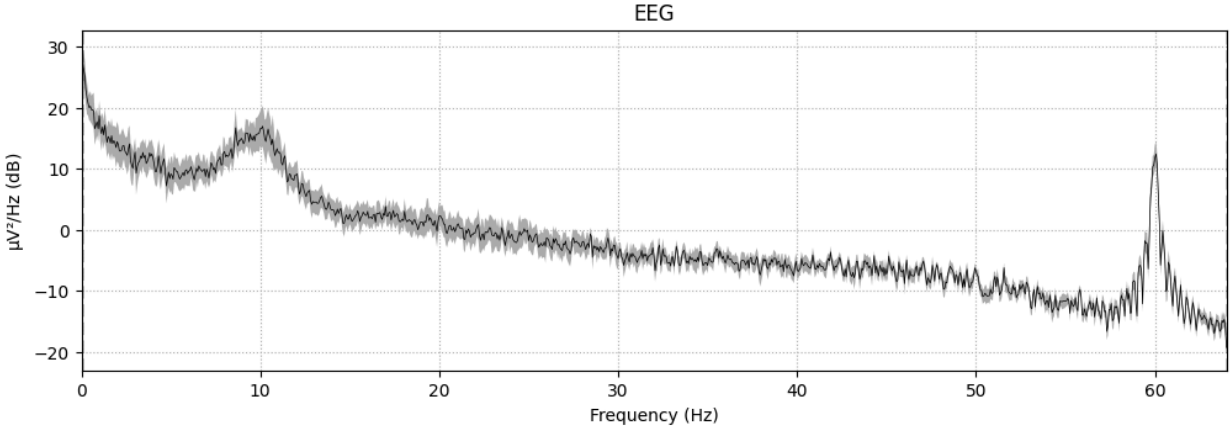


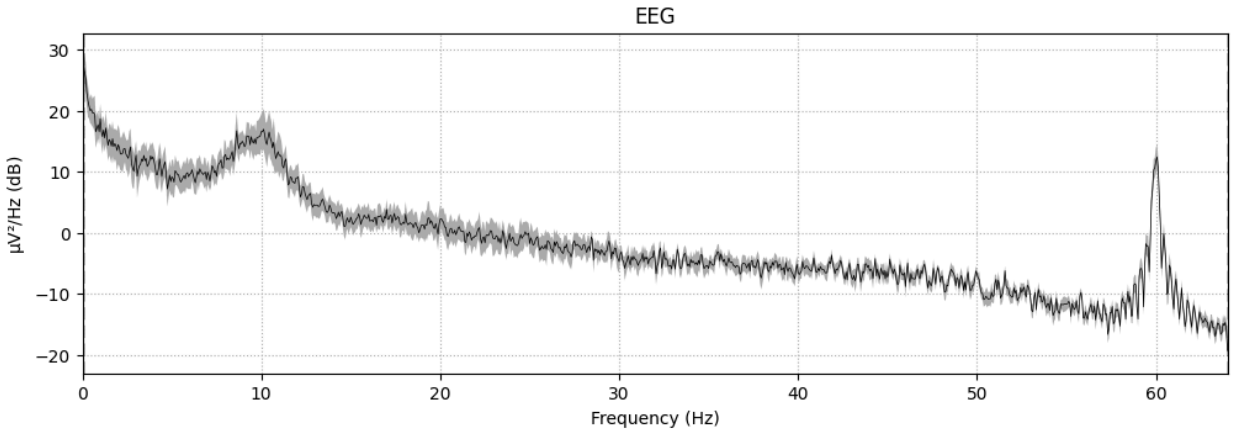
绘制原始数据功率谱图

In [8]:
```
raw.plot_psd(average=True)
```

```
NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plot().
Effective window size : 16.000 (s)
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/mne/viz/utils.py:165: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  (fig or plt).show(**kwargs)
```

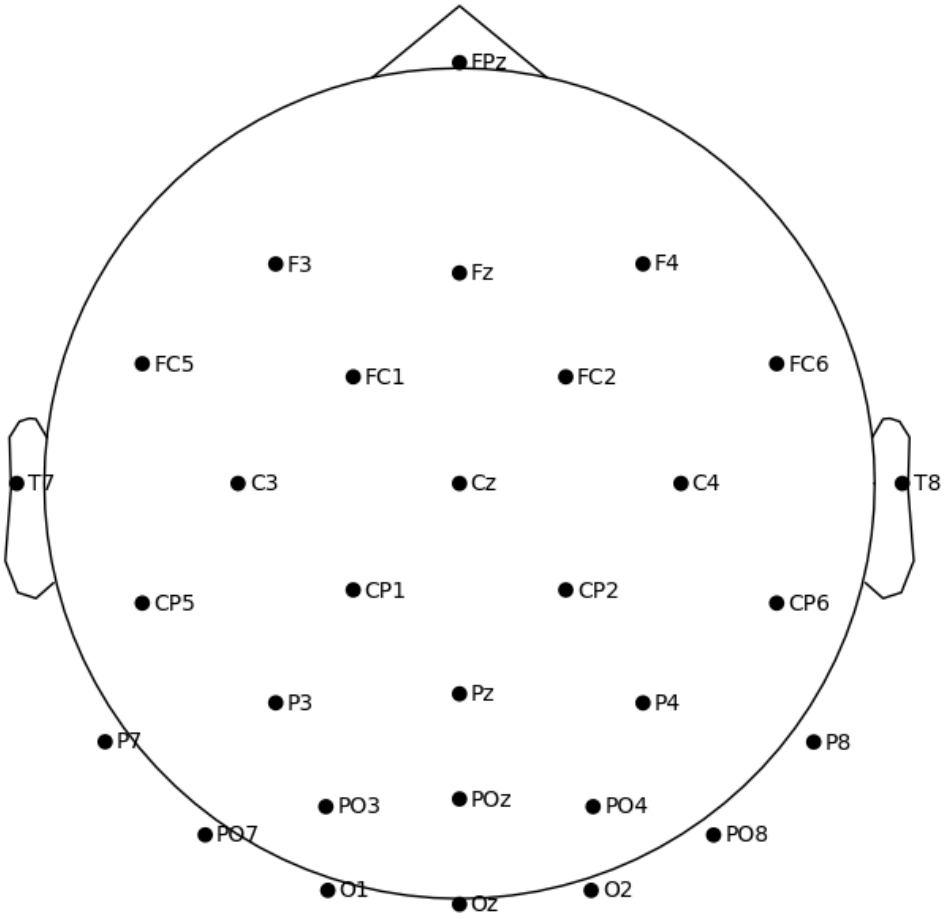
Out[8]:



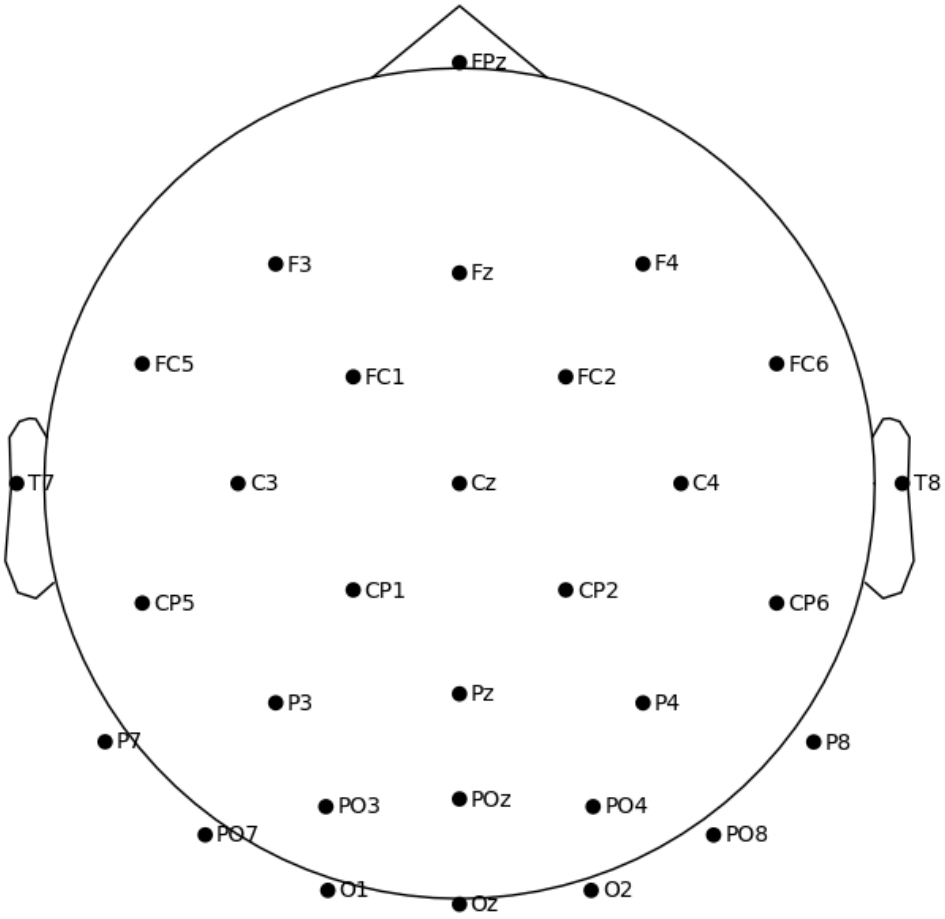


绘制导联空间位置图

In [9]:
```
raw.plot_sensors(ch_type='eeg', show_names=True)
```



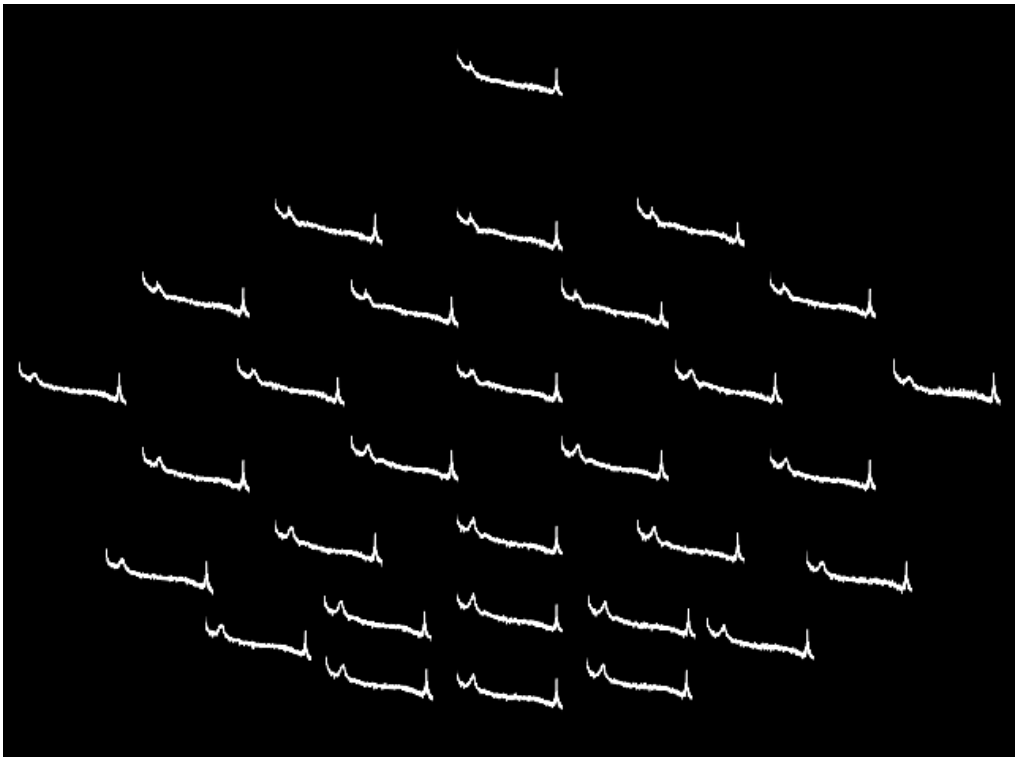
Out[9]:



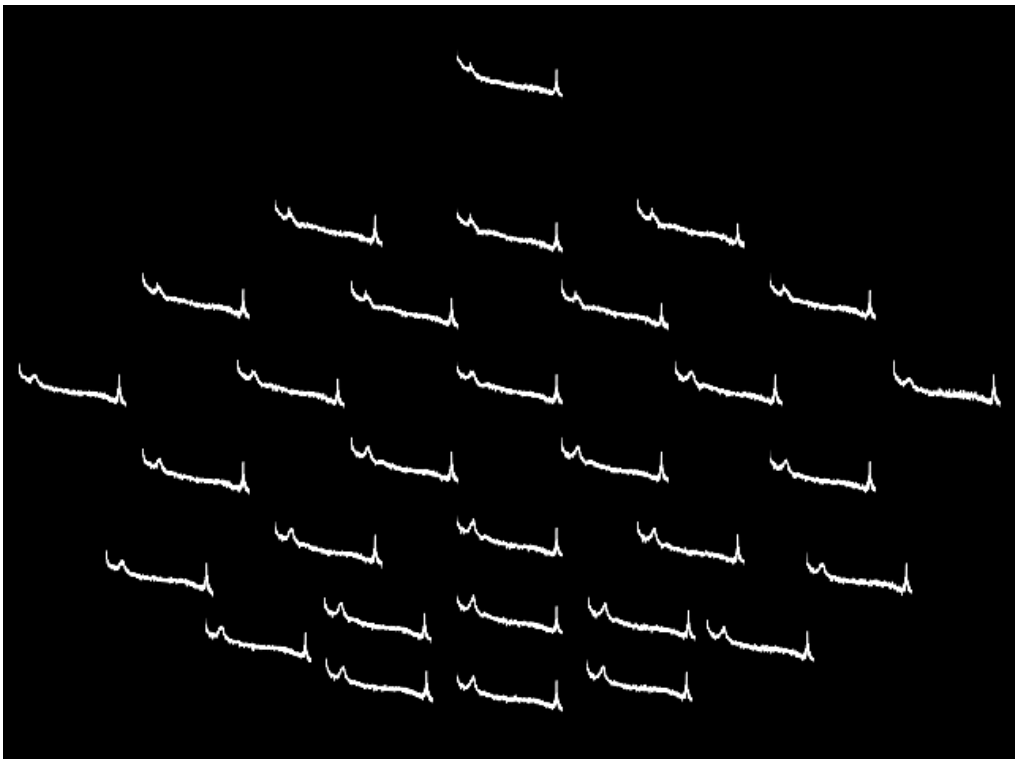
绘制拓扑图形式的原始数据功率谱图

In [10]:
```
raw.compute_psd().plot_topo()
```

```
Effective window size : 16.000 (s)
```



Out[10]:



第二步 - 滤波

陷波滤波

通过第一步中的功率谱图可以看到60Hz处可能存在环境噪音
这里首先使用陷波滤波器去掉工频
注意：在中国大陆及香港澳门地区（除台湾省以外）采集的数据一般工频会出现在50Hz处
此例比较例外，切记通过功率谱图判断

In [11]:
```
raw = raw.notch_filter(freqs=(60))
```

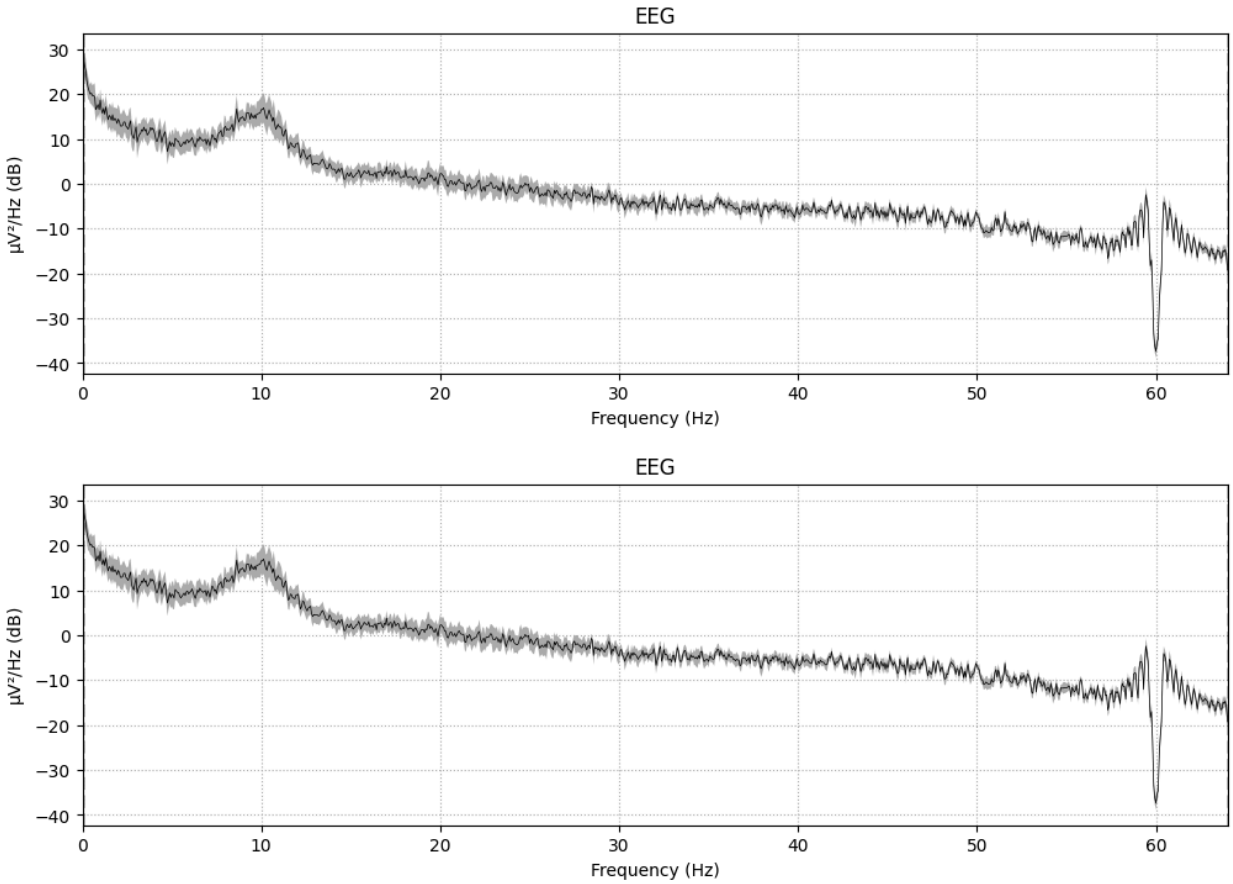
```
Filtering raw data in 1 contiguous segment
Setting up band-stop filter from 59 - 61 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal bandstop filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 59.35
- Lower transition bandwidth: 0.50 Hz (-6 dB cutoff frequency: 59.10 Hz)
- Upper passband edge: 60.65 Hz
- Upper transition bandwidth: 0.50 Hz (-6 dB cutoff frequency: 60.90 Hz)
- Filter length: 845 samples (6.602 s)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.1s finished
```

绘制功率谱图

In [12]:
```
raw.plot_psd(average=True)
```

```
NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plot().
Effective window size : 16.000 (s)
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/mne/viz/utils.py:165: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  (fig or plt).show(**kwargs)
```

Out[12]:



高/低通滤波

预处理步骤中，通常需要对数据进行高通滤波操作
此处采用最常规的滤波操作，进行30Hz的低通滤波及0.1Hz的高通滤波
高通滤波为了消除电压漂移，低通滤波为了消除高频噪音

In [13]:
```
raw = raw.filter(l_freq=0.1, h_freq=30)
```

```
Filtering raw data in 1 contiguous segment
Setting up band-pass filter from 0.1 - 30 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal bandpass filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 0.10
- Lower transition bandwidth: 0.10 Hz (-6 dB cutoff frequency: 0.05 Hz)
- Upper passband edge: 30.00 Hz
- Upper transition bandwidth: 7.50 Hz (-6 dB cutoff frequency: 33.75 Hz)
- Filter length: 4225 samples (33.008 s)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.3s finished
```

MNE中默认使用FIR滤波方法，若想使用IIR滤波方法，可通过修改参数method参数实现
默认method='fir'，使用IIR则修改为'iir'
对应代码即为：

> raw = raw.filter(l_freq=0.1, h_freq=30, method='iir')

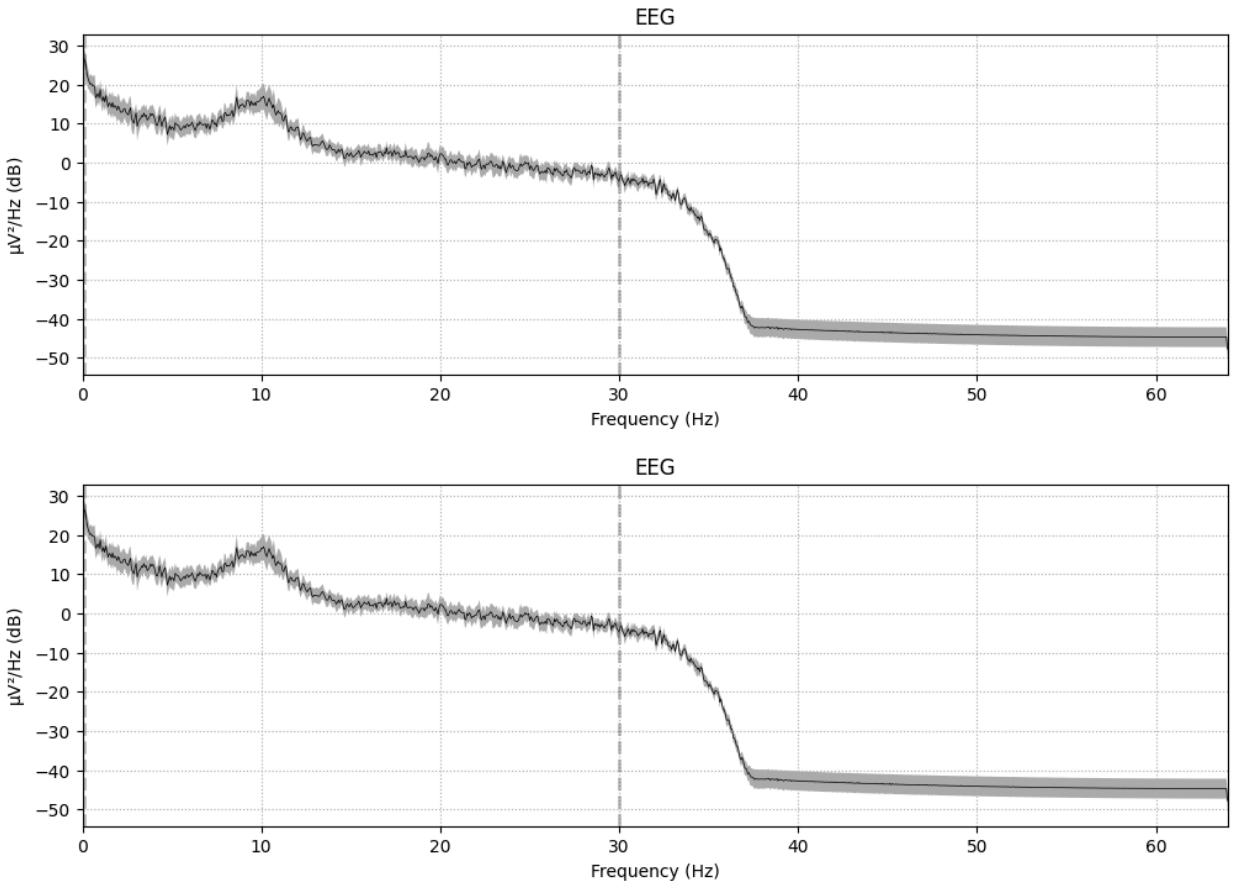
绘制功率谱图

In [14]:
```
raw.plot_psd(average=True)
```

```
NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plot().
Effective window size : 16.000 (s)
```

```
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/mne/viz/utils.py:165: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  (fig or plt).show(**kwargs)
```

Out[14]:



第三步 - 去伪迹

去坏段

MNE中可以通过打开交互式数据地形图界面，手动进行坏段标记

```
fig = raw.plot()
fig.fake_keypress('a')
```

按a就可以打开这个GUI小窗口，add new label可以添加一个用于标记坏段的marker
在MNE中，并不会将坏段直接删掉，而是进行了数据标记
在之后的数据处理中，
将进行数据处理的函数中的参数reject_by_annotation设为True即可在处理过程中自动排除标记的片段
如果遇到GUI窗口无法弹出，需在脚本开头添加如下代码：

```
import matplotlib
matplotlib.use('TkAgg')
```

注意：不推荐在Jupyter notebook中打开，大概率卡死

去坏道

MNE中坏的导联也不是直接删掉，也是通过对坏道进行'bads'标记
在这个例子中，假定导联'FC5'为坏道，则把'FC5'进行坏道标记

In [15]:

```
# 标记坏道
raw.info['bads'].append('FC5')
# 打印当前坏道
print(raw.info['bads'])
```

```
['FC5']
```

当然，也可以添加多个坏道
如若'FC5'和'C3'都为坏道，则通过下述代码标记：

> raw.info['bads'].extend(['FC5', 'C3'])

坏道插值重构

MNE的坏道重建即是对标记为'bads'的导联进行了信号重建

In [16]:
```
raw = raw.interpolate_bads()
```

```
Setting channel interpolation method to {'eeg': 'spline'}.
Interpolating bad channels.
    Automatic origin fit: head of radius 95.0 mm
Computing interpolation matrix from 29 sensor positions
Interpolating 1 sensors
```

进行信号重建后会默认把坏掉的'bads'标记去掉
如果不想去掉对原坏道的标记，则将reset_bads参数设为False即可，
对应代码如下：

> raw = raw.interpolate_bads(reset_bads=False)

独立成分分析 （Independent Components Analysis，ICA）

运行ICA

MNE中进行ICA的编程思路是首先构建一个ICA对象（可以理解成造一个ICA分析器）
然后用这个ICA分析器对脑电数据进行分析（通过ICA对象的一系列方法）
由于ICA对低频分离效果不好
这里对高通1Hz的数据进行ICA及相关成分剔除，再应用到高通0.1Hz的数据上

In [17]:
```
ica = ICA(max_iter='auto')
raw_for_ica = raw.copy().filter(l_freq=1, h_freq=None)
ica.fit(raw_for_ica)
```

```
Filtering raw data in 1 contiguous segment
Setting up high-pass filter at 1 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal highpass filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 1.00
- Lower transition bandwidth: 1.00 Hz (-6 dB cutoff frequency: 0.50 Hz)
- Filter length: 423 samples (3.305 s)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:    0.0s
Fitting ICA to data using 30 channels (please be patient, this may take a while)
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.5s finished
Selecting by non-zero PCA components: 29 components
Fitting ICA took 4.6s.
```

Out[17]:

Method	fastica
Fit parameters	algorithm=parallel fun=logcosh fun_args=None max_iter=1000
Fit	117 iterations on raw data (30504 samples)
ICA components	29
Available PCA components	30
Channel types	eeg
ICA components marked for exclusion	—

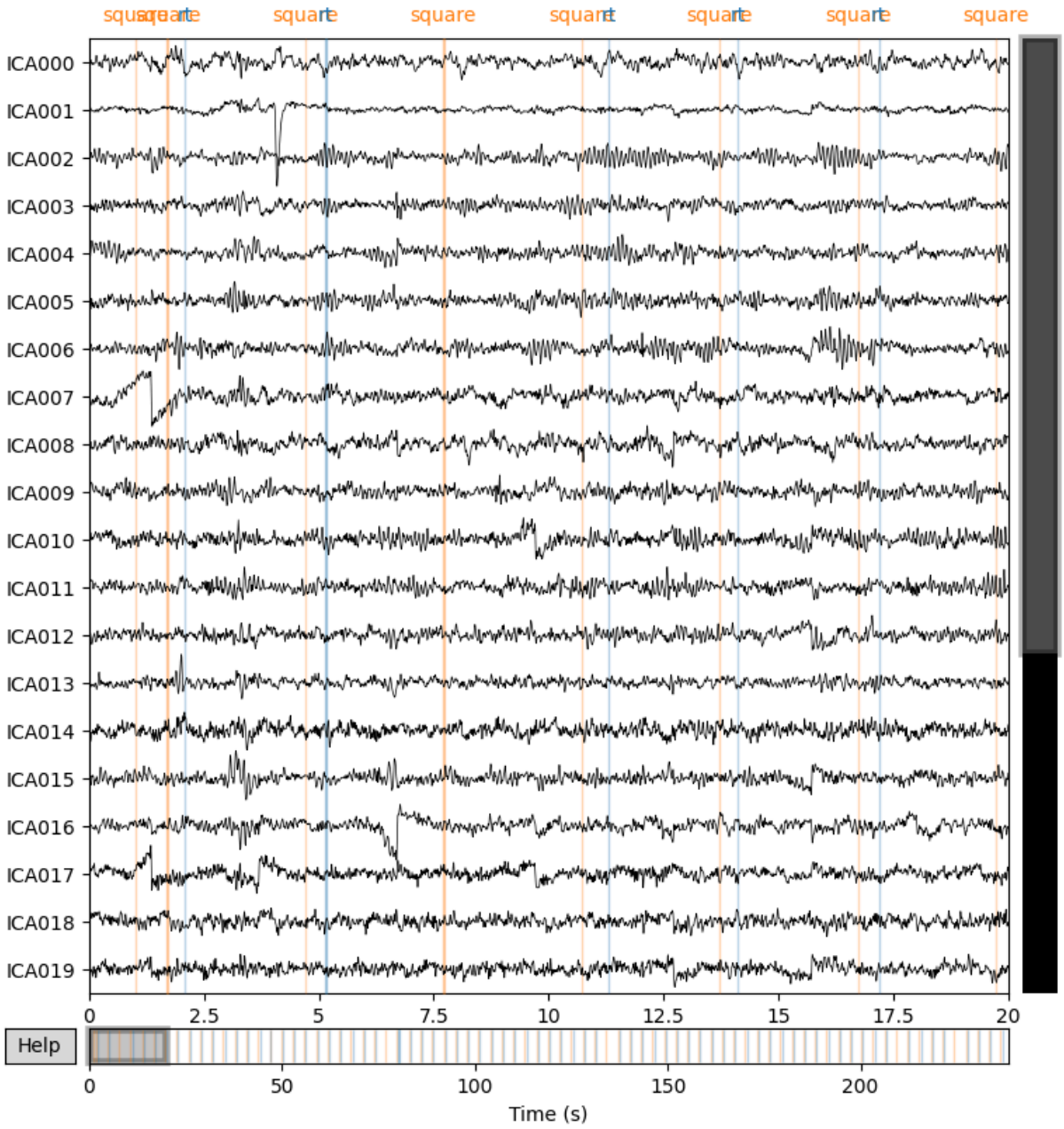
这里没有设定n_components，即ICA的成分数让MNE的ICA分析器自动去选择
类似EEGLAB，如果希望ICA的成分数为固定个数，可以自定义设置（n_components<=n_channels）
以30个独立成分为例，对应代码改为如下即可：

> ica = ICA(n_components=30, max_iter='auto')

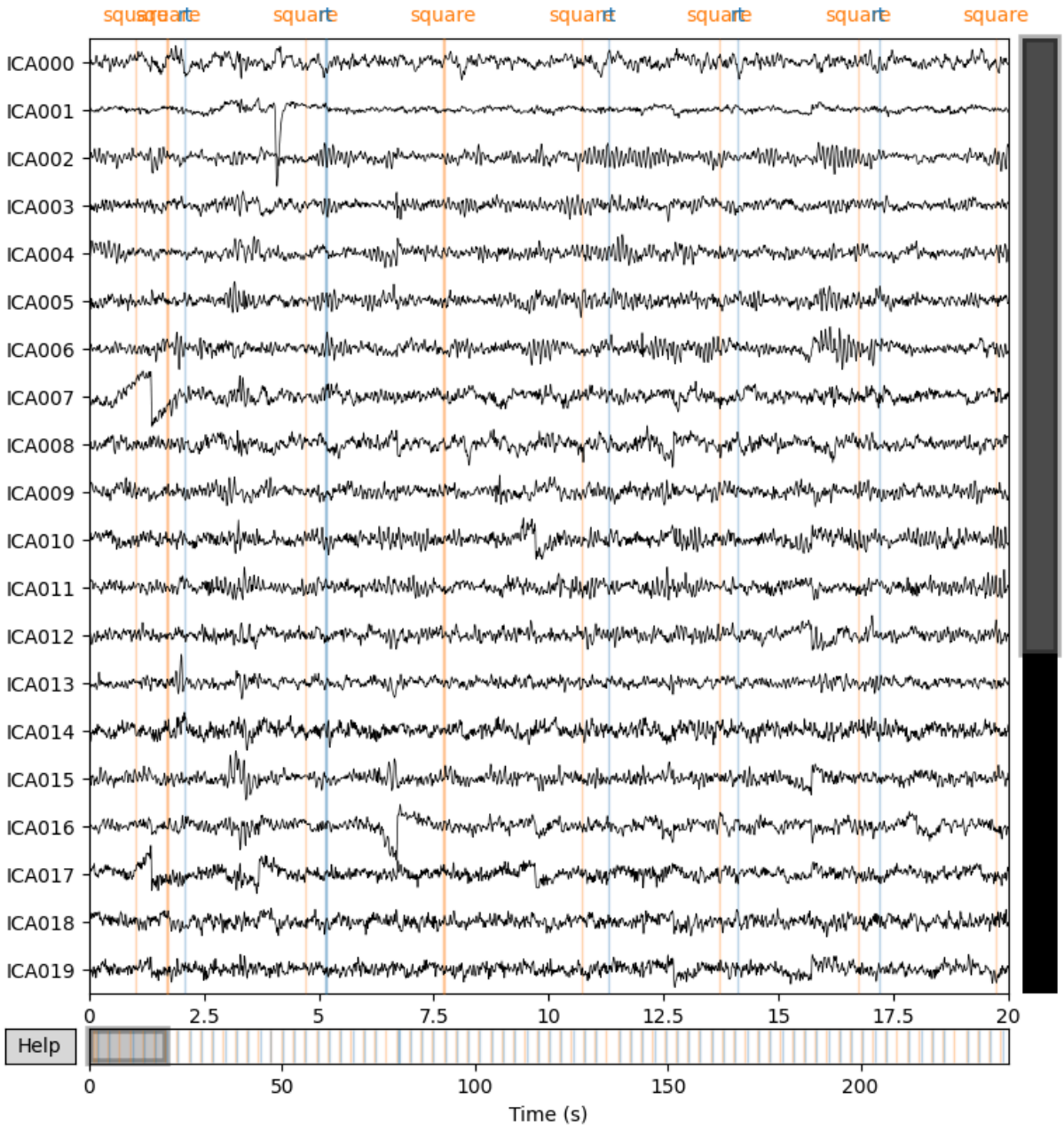
绘制各独立成分的时序信号图

In [18]:
```
ica.plot_sources(raw_for_ica)
```

```
Creating RawArray with float64 data, n_channels=31, n_times=30504
    Range : 0 ... 30503 =      0.000 ...   238.305 secs
Ready.
```

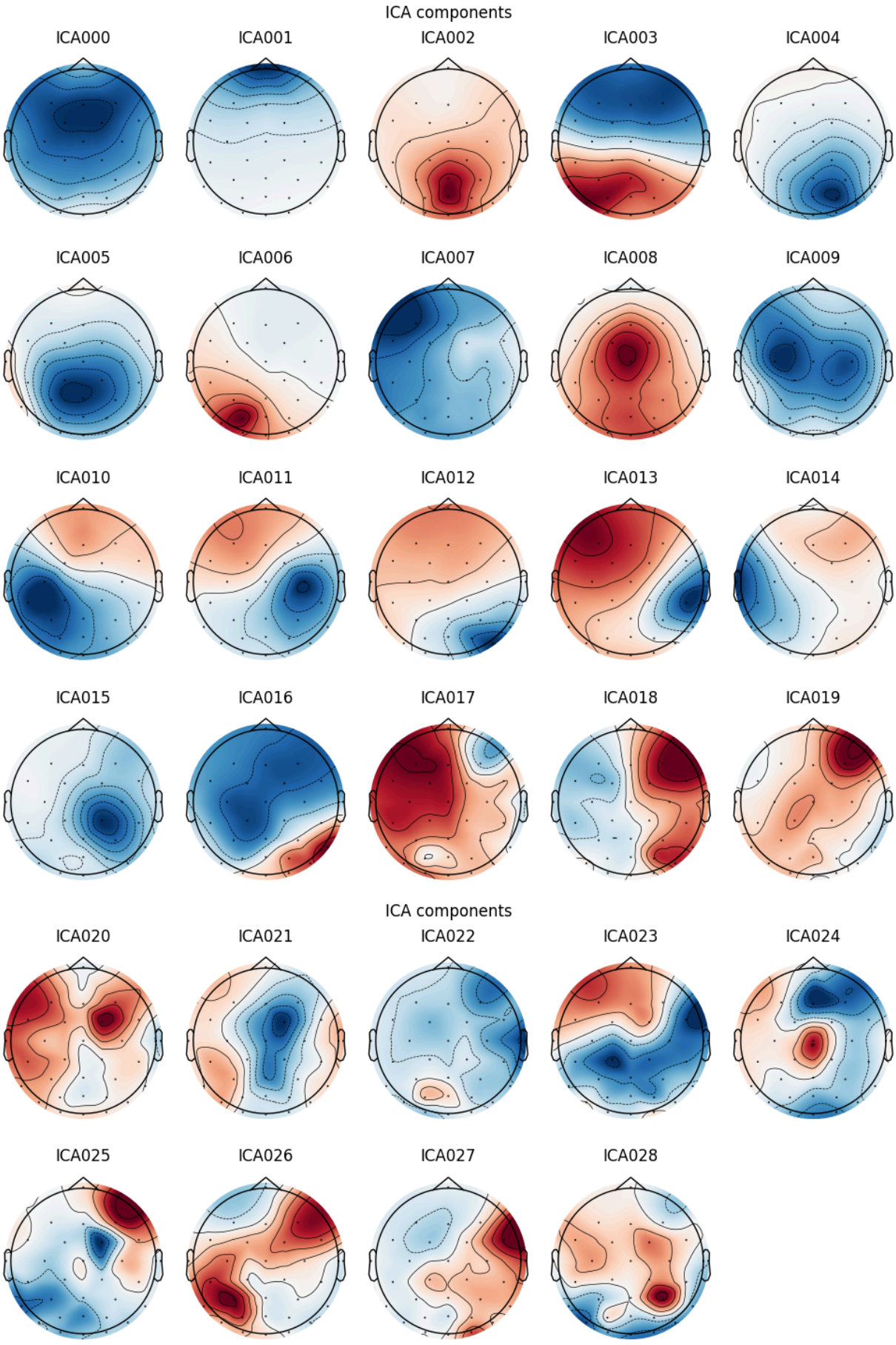


Out[18]:



绘制各成分的地形图

In [19]:
```
ica.plot_components()
```



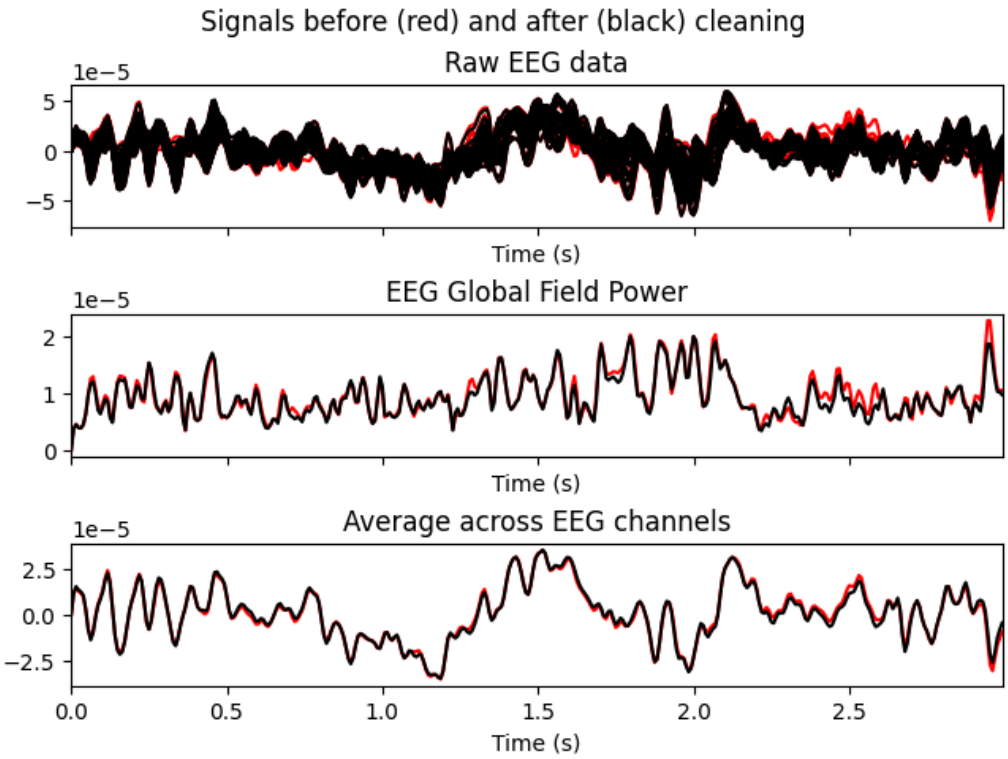
Out[19]: [<MNEFigure size 975x967 with 20 Axes>, <MNEFigure size 975x496 with 9 Axes>]

查看去掉某一或某些成分前后的信号差异

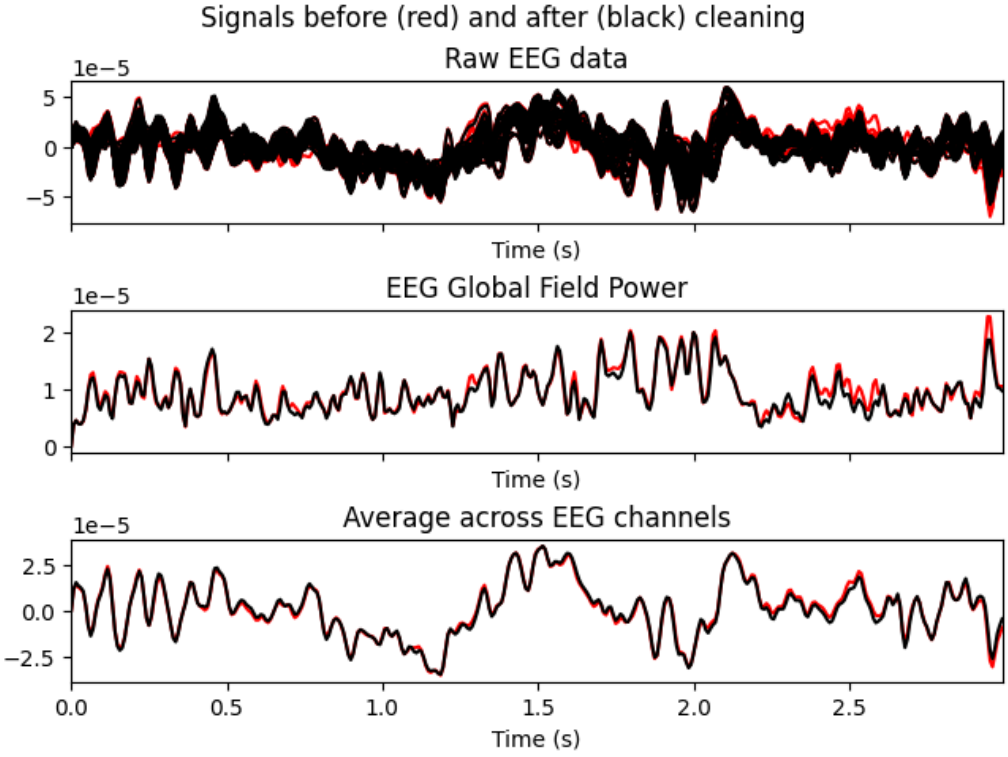
这里以去掉第2个成分（即ICA001）为例

In [20]:
```
ica.plot_overlay(raw_for_ica, exclude=[1])
```

```
Applying ICA to Raw instance
    Transforming to ICA space (29 components)
    Zeroing out 1 ICA component
    Projecting back using 30 PCA components
```



Out[20]:

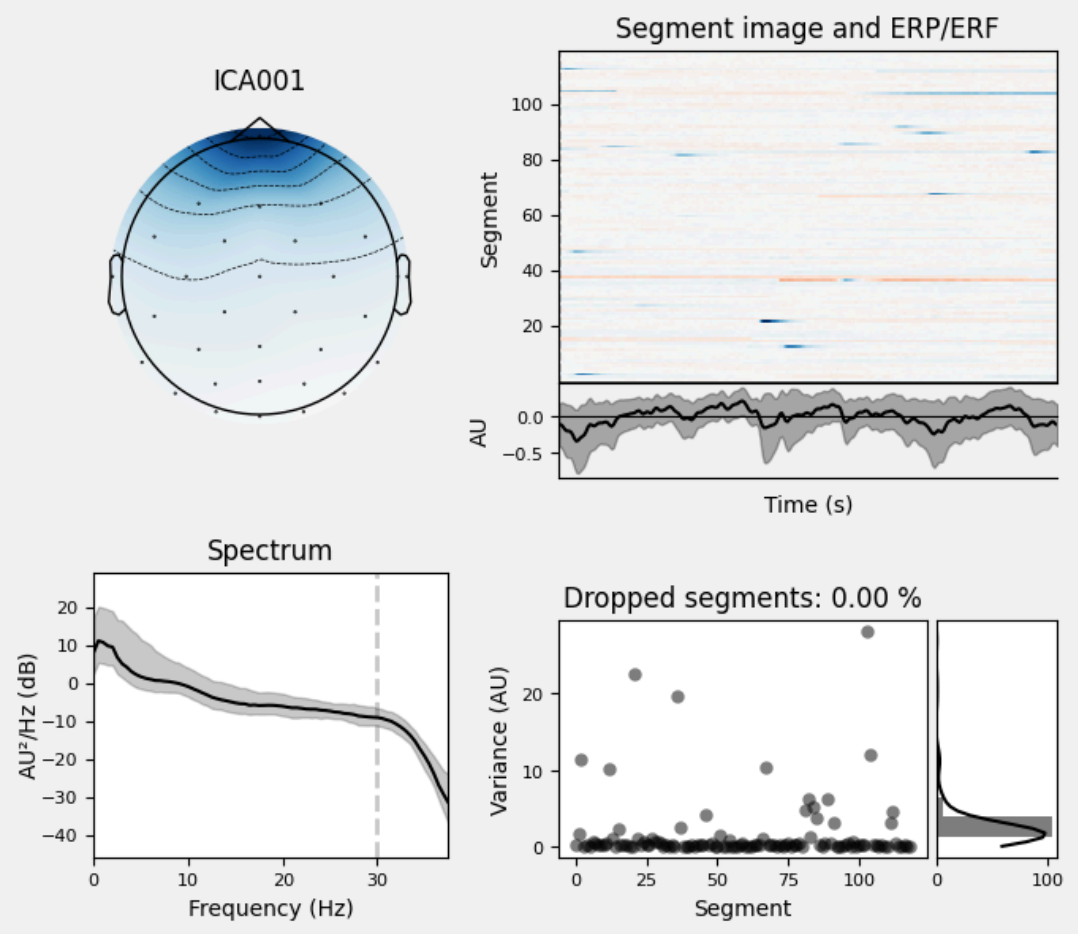


可视化单个独立成分的信息

这里可视化第2个成分（ICA001）

In [21]:
```
ica.plot_properties(raw, picks=[1])
```

```
    Using multitaper spectrum estimation with 7 DPSS windows
Not setting metadata
119 matching events found
No baseline correction applied
0 projection items activated
```



Out[21]: [<Figure size 700x600 with 6 Axes>]

成分ICA001的能量较高处为前额，且在低频处能量较高，在一些trials中有明显增强
可以判断为一个眼动成分

剔除成分

In [22]:
```
# 设定要删除的成分序号
ica.exclude = [1]
# 应用到脑电数据上
ica.apply(raw)
```

```
Applying ICA to Raw instance
    Transforming to ICA space (29 components)
    Zeroing out 1 ICA component
    Projecting back using 30 PCA components
```

Out[22]:

▼ General

Measurement date	Unknown
Experimenter	Unknown
Participant	Unknown

▼ Channels

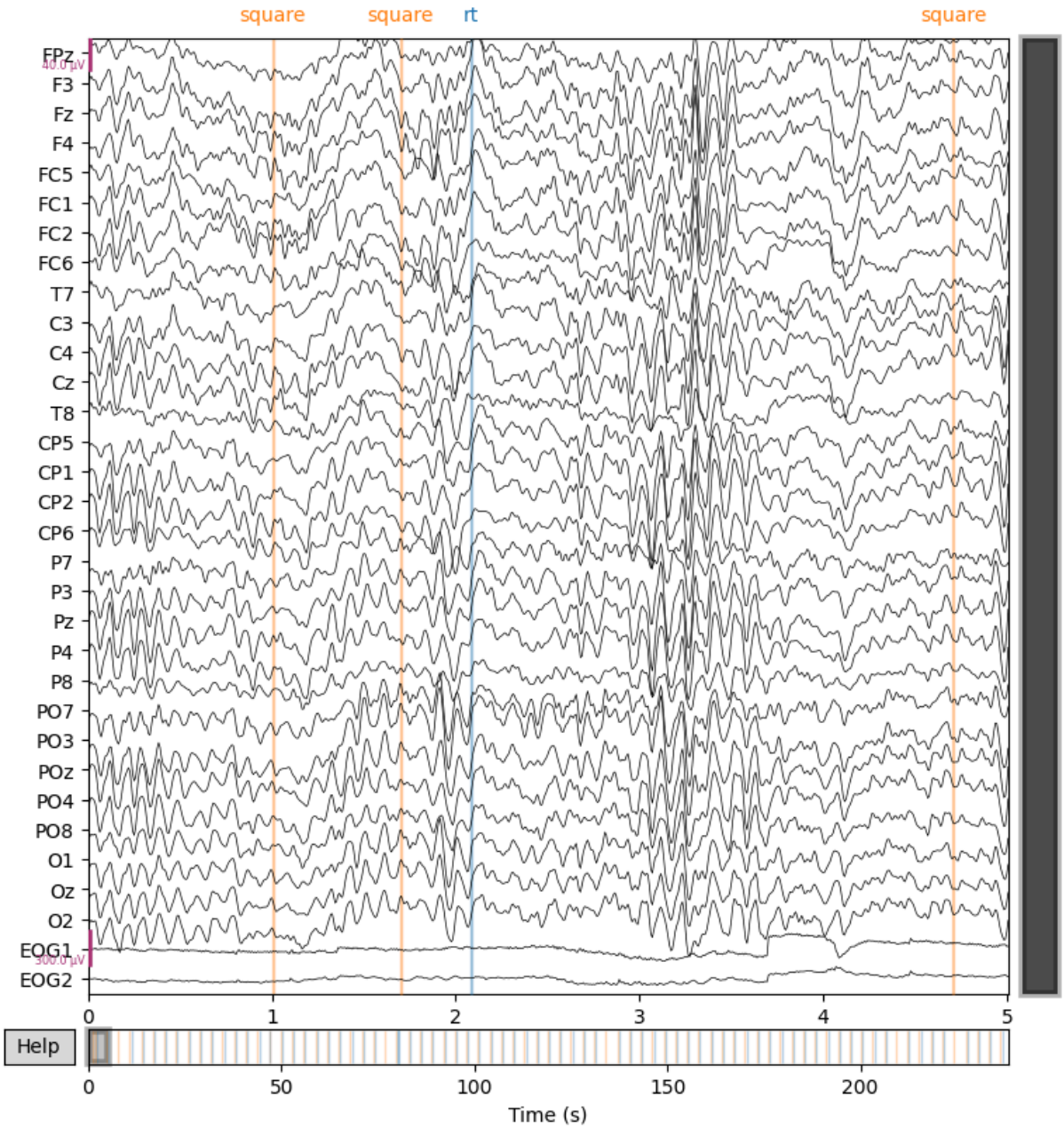
Digitized points	35 points
Good channels	30 EEG, 2 EOG
Bad channels	None
EOG channels	EOG1, EOG2
ECG channels	Not available

▼ Data

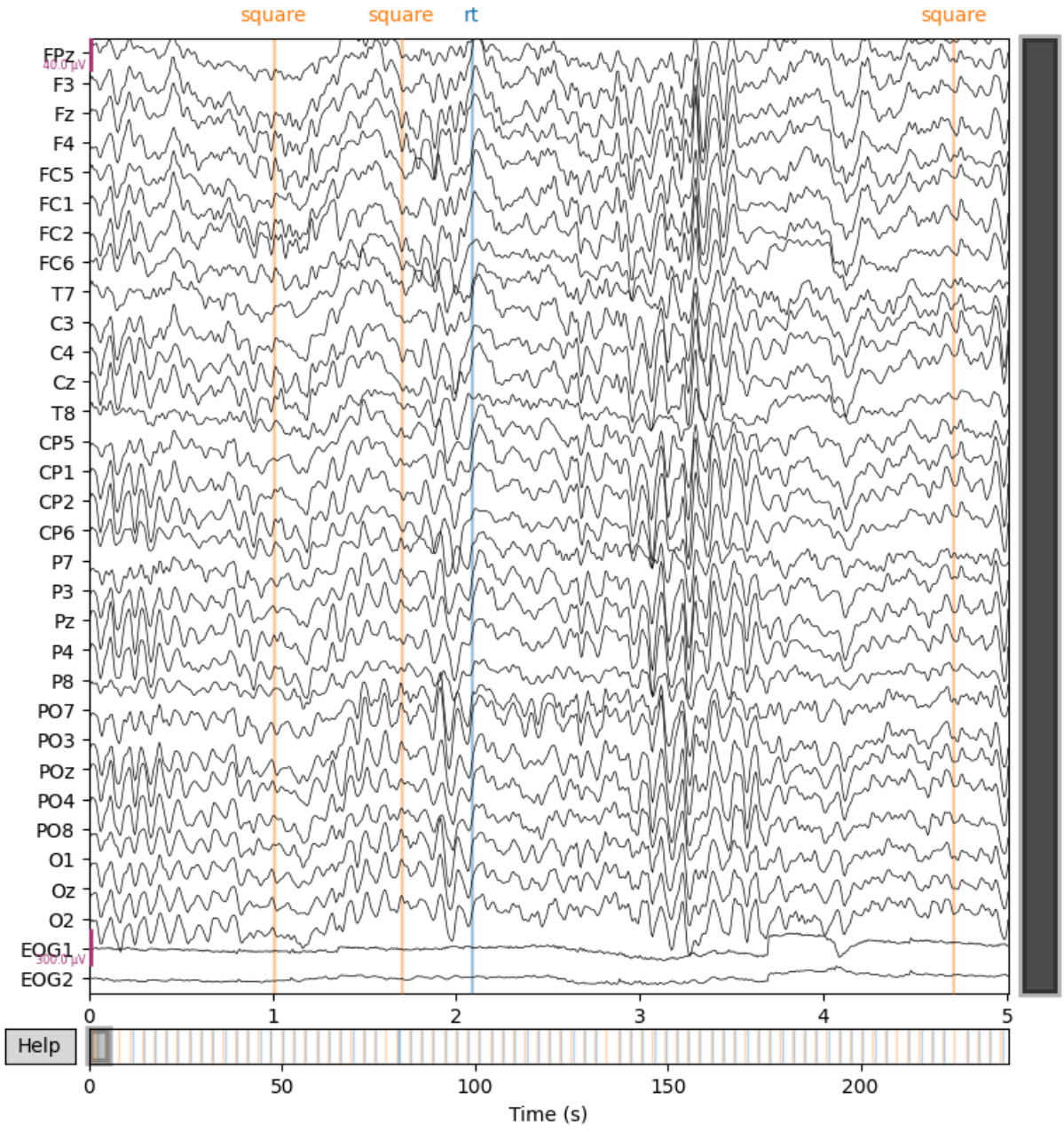
Sampling frequency	128.00 Hz
Highpass	0.10 Hz
Lowpass	30.00 Hz
Filenames	eeglab_data.fdt
Duration	00:03:59 (HH:MM:SS)

绘制ICA后的数据波形图

In [23]:
```
raw.plot(duration=5, n_channels=32, clipping=None)
```


square
square
rt
square
FPz
F3
Fz
F4
FC5
FC1
FC2
FC6
T7
C3
C4
Cz
T8
CP5
CP1
CP2
CP6
P7
P3
Pz
P4
P8
PO7
PO3
POz
PO4
PO8
O1
Oz
O2
EOG1
EOG2
40.0 µV
300.0 µV
0
1
2
3
4
5
Help
0
50
100
150
200
Time (s)

Out[23]:



第四步 - 重参考

若数据需要进行参考，以'TP9'和'TP10'为参考电极为例，可以使用以下代码：

```
raw.set_eeg_reference(ref_channels=['TP9','TP10'])
```

若使用平均参考，则使用以下代码：

```
raw.set_eeg_reference(ref_channels='average')
```

若使用REST参考，则使用以下代码：
这里需要传入一个forward参数，详情可参考MNE对应介绍：
https://mne.tools/stable/auto_tutorials/preprocessing/55_setting_eeg_reference.html

```
raw.set_eeg_reference(ref_channels='REST', forward=forward)
```

若使用双极参考，则使用以下代码：（这里'EEG X'和'EEG Y'分别对应用于参考的阳极和阴极导联)

```
raw_bip_ref = mne.set_bipolar_reference(raw, anode=['EEG X'], cathode=['EEG Y'])
```

第五步 - 数据分段

提取事件信息

首先，需要确定分段需要用到的markers
查看数据中的markers：

In [24]:
```
print(raw.annotations)
```

```
<Annotations | 154 segments: rt (74), square (80)>
```

即数据中包含两种markers，分别为'square'和'rt'

MNE有两种数据结构存储事件信息，分别为Events和Annotations

对于Annotations对象，其用字符串来表示时间类型，如上打印出来的所示
其用时间点表示时间，且包含marker的持续时长，当然瞬时marker其持续时长为0
其内部数据表示为一个类似List的类

In [25]:
```
# 基于Annotations打印数据的事件持续时长

print(raw.annotations.duration)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [26]:
```
# 基于Annotations打印数据的事件的描述信息

print(raw.annotations.description)
```

```
['square' 'square' 'rt' 'square' 'rt' 'square' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square'
 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square'
 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square'
 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'square' 'rt' 'square'
 'rt' 'square' 'rt' 'square' 'rt' 'square' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt']
```

In [27]:
```
# 基于Annotations打印数据的事件的开始时间

print(raw.annotations.onset)
```

```
[  1.000068   1.695381   2.082407   4.703193   5.148224   7.711006
  10.718818  11.303858  13.726631  14.116658  16.734443  17.187474
  19.742256  20.199287  22.750068  23.136095  25.757881  26.124906
  28.765693  29.140719  31.773506  32.116529  34.781318  35.24635
  37.789131  38.254163  40.796943  41.229973  43.804756  44.30079
  46.812568  47.156592  49.820381  50.273412  52.828193  53.242222
  55.836006  56.211031  58.843818  59.237845  61.851631  62.183654
  64.859443  65.257471  67.867256  68.598306  70.875068  71.277096
  73.882881  74.323911  76.890693  79.898506  80.269531  82.906318
  83.312346  85.914131  86.313158  88.921943  89.280968  91.929756
  92.323783  94.937568  95.296593  97.945381  98.37141  100.953193
 101.363221 103.961006 104.332031 106.968818 107.343844 109.976631
 110.425662 112.984443 113.410472 115.992256 116.367281 119.000068
 119.359093 122.007881 122.409908 125.015693 125.441722 128.023506
 128.422533 131.031318 131.433346 134.039131 137.046943 137.488974
 140.054756 140.445783 143.062568 143.507599 146.070381 146.527412
 149.078193 149.504222 152.086006 152.465032 155.093818 155.479845
 158.101631 158.452655 161.109443 161.515471 164.117256 164.46928
 167.125068 167.543097 170.132881 170.507906 173.140693 173.636727
 176.148506 176.597537 179.156318 179.62535  182.164131 182.59016
 185.171943 185.678978 188.179756 188.585784 191.187568 191.675602
 194.195381 194.551405 197.203193 197.707228 200.211006 200.609033
 203.218818 203.667849 206.226631 206.65566  209.234443 212.242256
 212.621282 215.250068 215.641095 218.257881 218.667909 221.265693
 221.617717 224.273506 227.281318 227.718348 230.289131 230.71116
 233.296943 233.729973 236.304756 236.753787]
```

而Events对象，则是数据分段需要用到的一种事件记录数据类型
其用一个整型'Event ID'编码事件类型，以样本的形式来表示时间
且不含有marker的持续时长，其内部数据类型为NumPy Array

事件信息数据类型转换

将Annotations类型的事件信息转为Events类型

In [28]:
```
events, event_id = mne.events_from_annotations(raw)
```

```
Used Annotations descriptions: ['rt', 'square']
```

‘events’为记录时间相关的矩阵，‘event_id’为不同markers对应整型的字典信息
这里打印出events矩阵的shape和event_id

In [29]:
```
print(events.shape, event_id)
```

```
(154, 3) {'rt': 1, 'square': 2}
```

即'rt' marker对应数字1，'square' marker对应数字2
共154个markers

数据分段

基于Events对数据进行分段
这里提取刺激前1秒到刺激后2秒的数据，即'square' marker对应-1s到2s的数据
取baseline时间区间为刺激前0.5s到刺激出现
并进行卡阈值，即在epoch中出现最大幅值与最小幅值的差大于2×10^{-4}则该epoch被剔除
注意：这里的阈值设置较大，一般数据质量佳的情况下推荐设置为5×10^{-5}到1×10^{-4}之间

In [30]:
```
epochs = mne.Epochs(raw, events, event_id=2, tmin=-1, tmax=2, baseline=(-0.5, 0),
                    preload=True, reject=dict(eeg=2e-4))
```

```
Not setting metadata
80 matching events found
Applying baseline correction (mode: mean)
0 projection items activated
Using data from preloaded Raw for 80 events and 385 original time points ...
0 bad epochs dropped
```

即分段后的数据存为了Epochs类的对象epochs
打印epochs即可看到分段后数据的相关信息

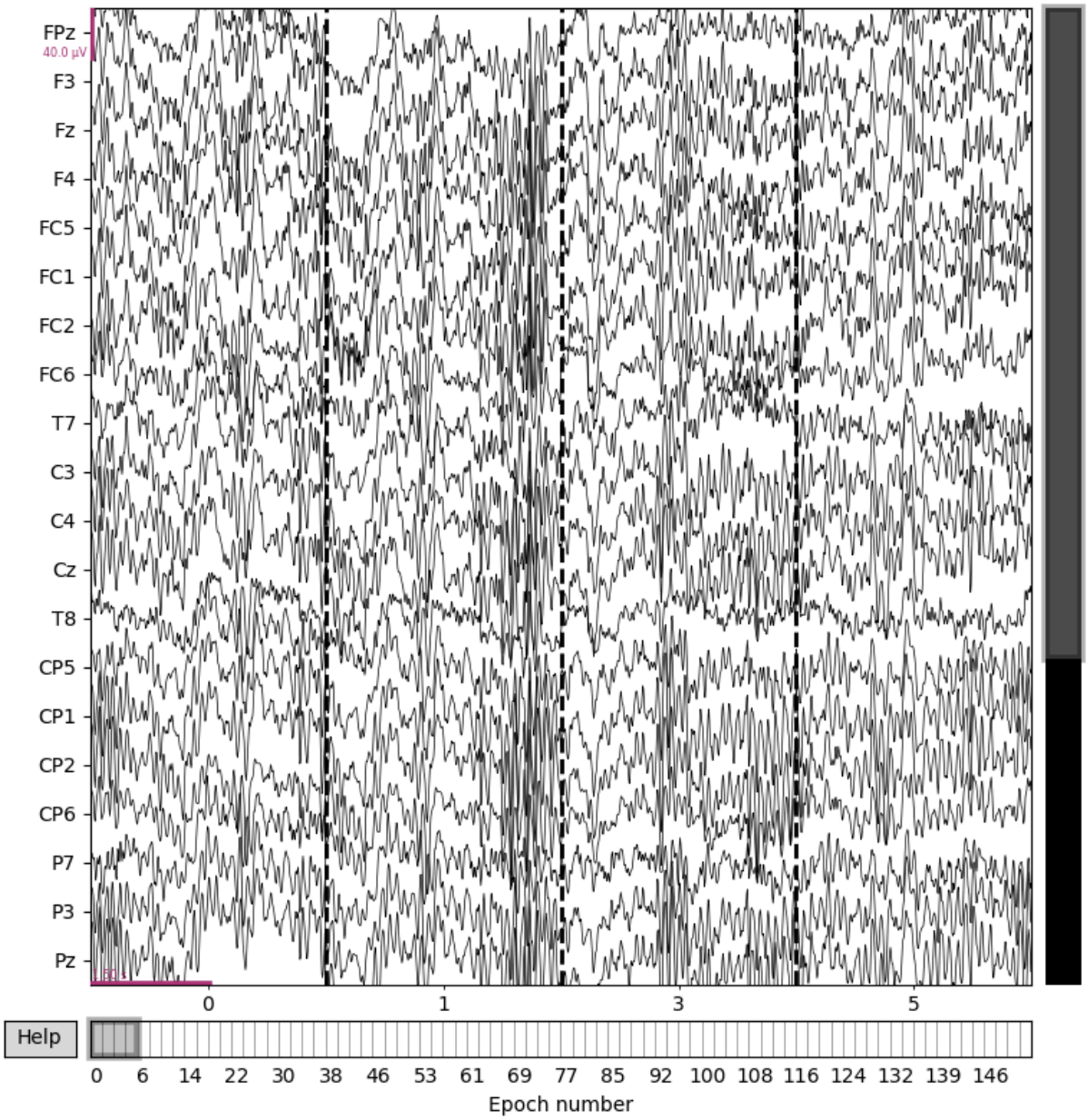
In [31]:
```
print(epochs)
```

```
<Epochs |  80 events (all good), -1 – 2 s, baseline -0.5 – 0 s, ~7.6 MB, data loaded,
 '2': 80>
```

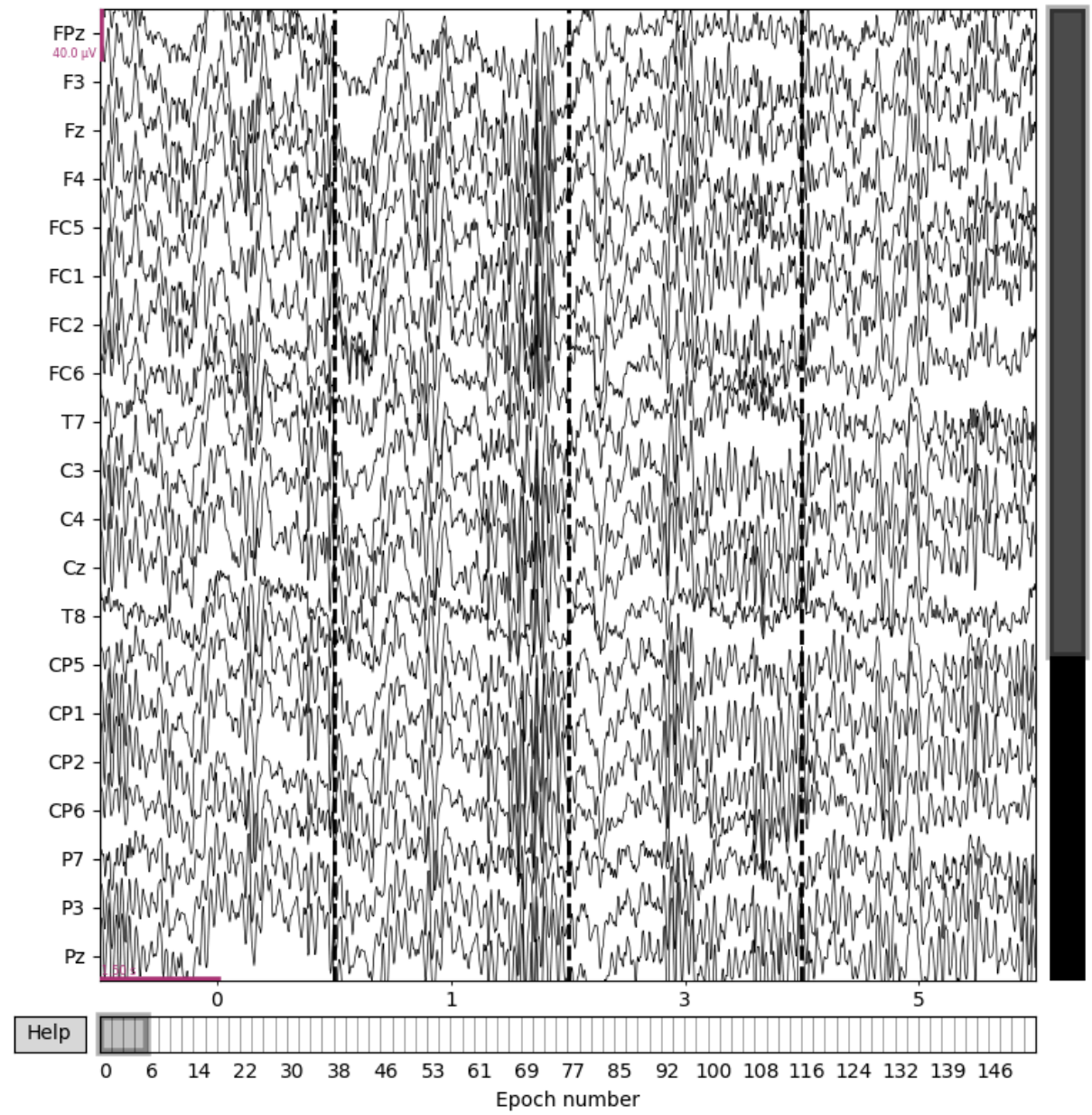
可视化分段后的数据

可视化分段数据（这里显示4个epochs）

In [32]:
```
epochs.plot(n_epochs=4)
```



Out[32]:

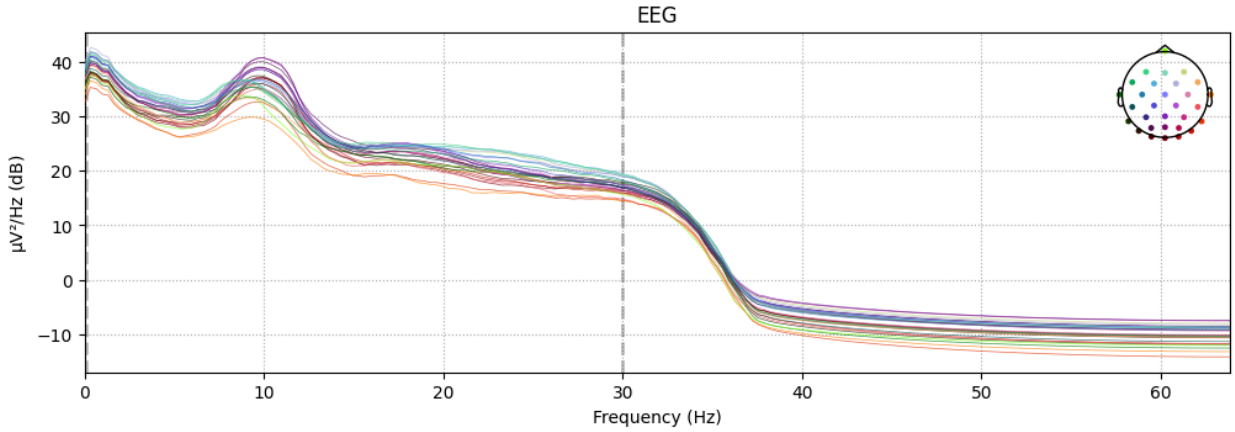


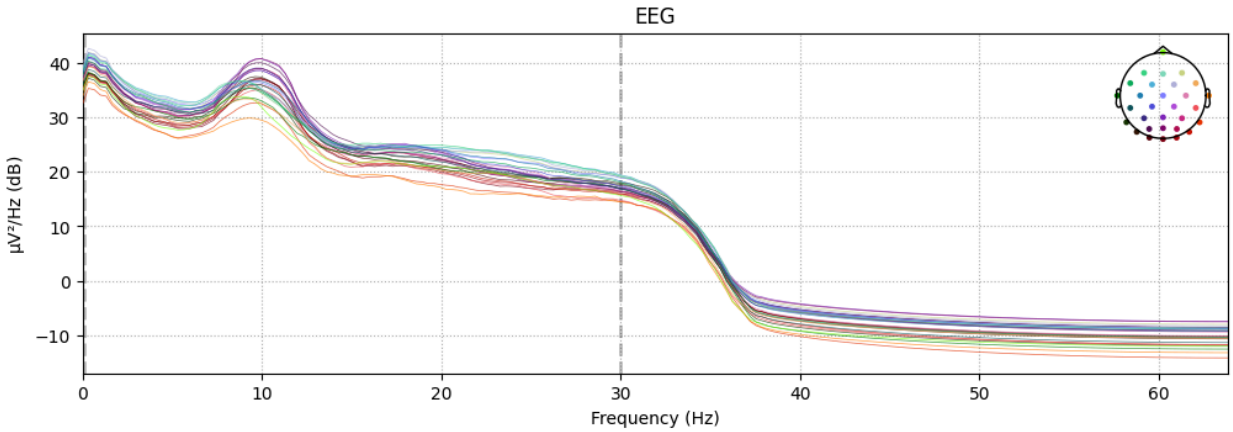
绘制功率谱图（逐导联）

In [33]:
```
epochs.compute_psd().plot(picks='eeg')
```

```
    Using multitaper spectrum estimation with 7 DPSS windows
Averaging across epochs...
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/mne/viz/utils.py:165: UserWarning: Matplotlib
is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  (fig or plt).show(**kwargs)
```

Out[33]:

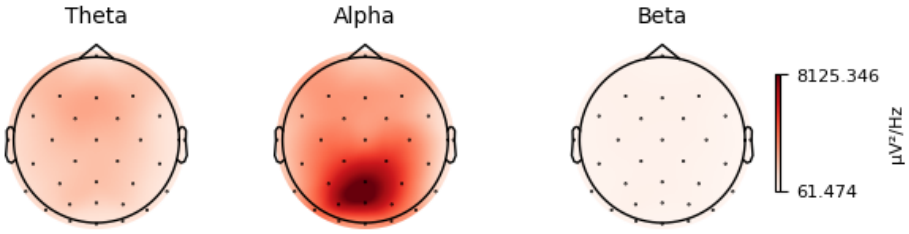




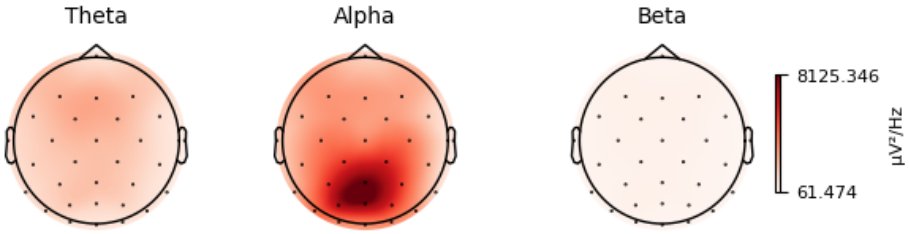
绘制功率谱拓扑图（分Theta、Alpha和Beta频段）

In [34]:
```
bands = [(4, 8, 'Theta'), (8, 12, 'Alpha'), (12, 30, 'Beta')]
epochs.plot_psd_topomap(bands=bands, vlim='joint')
```

```
NOTE: plot_psd_topomap() is a legacy function. New code should use .compute_psd().plot_topomap().
    Using multitaper spectrum estimation with 7 DPSS windows
converting legacy list-of-tuples input to a dict for the `bands` parameter
```



Out[34]:



第六步 - 叠加平均

MNE中使用Epochs类来存储分段数据，用Evoked类来存储叠加平均数据

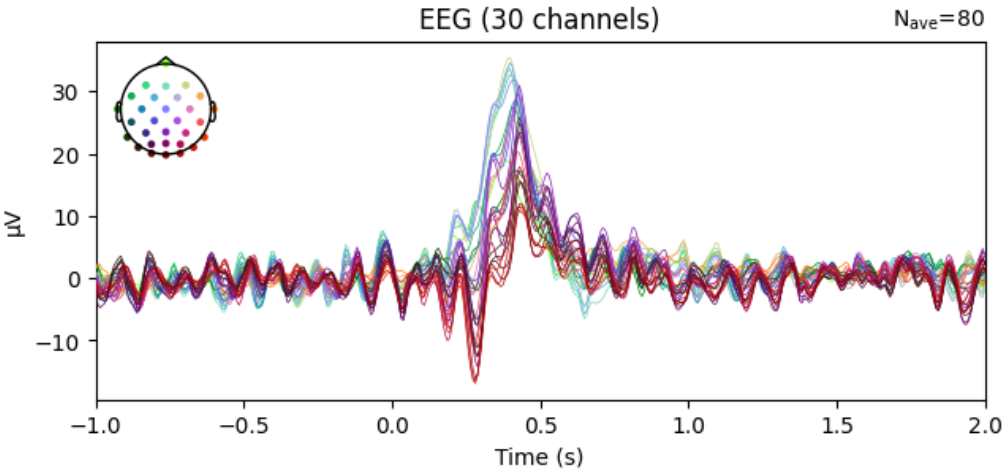
数据叠加平均

In [35]:
```
evoked = epochs.average()
```

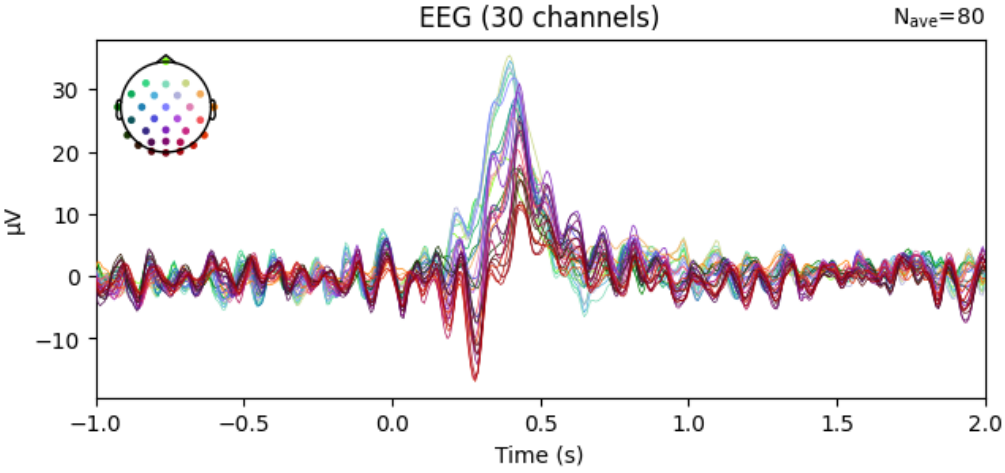
可视化叠加平均后的数据

绘制逐导联的时序信号图

In [36]:
```
evoked.plot()
```

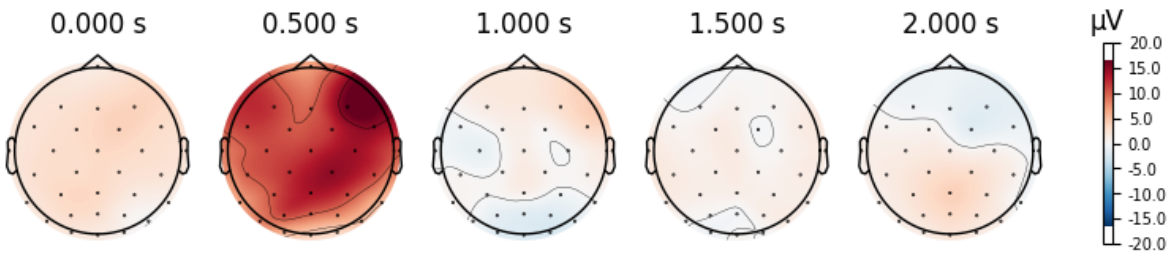


Out[36]:

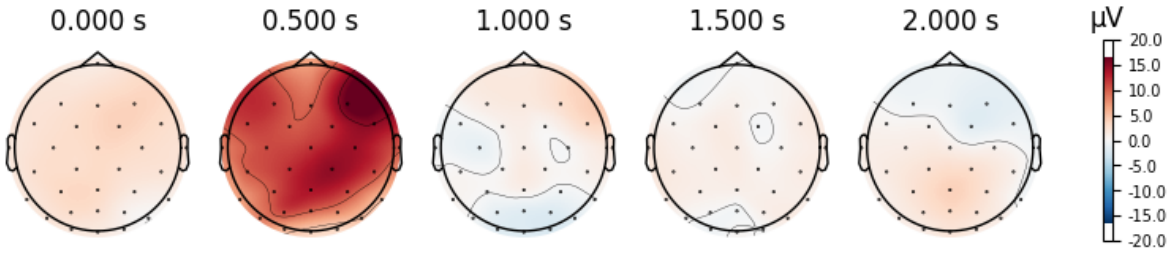


绘制地形图

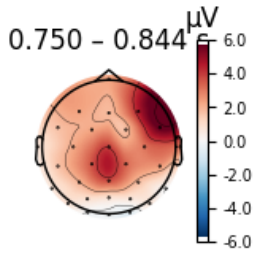
In [37]:
```
# 绘制0ms、0.5s、1s、1.5s和2s处的地形图
times = np.linspace(0, 2, 5)
evoked.plot_topomap(times=times, colorbar=True)
```



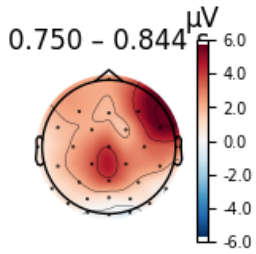
Out[37]:



In [38]:
```
# 绘制某一特定时刻的地形图
# 此例绘制0.8s处，取0.75-0.85s的均值

evoked.plot_topomap(times=0.8, average=0.1)
```



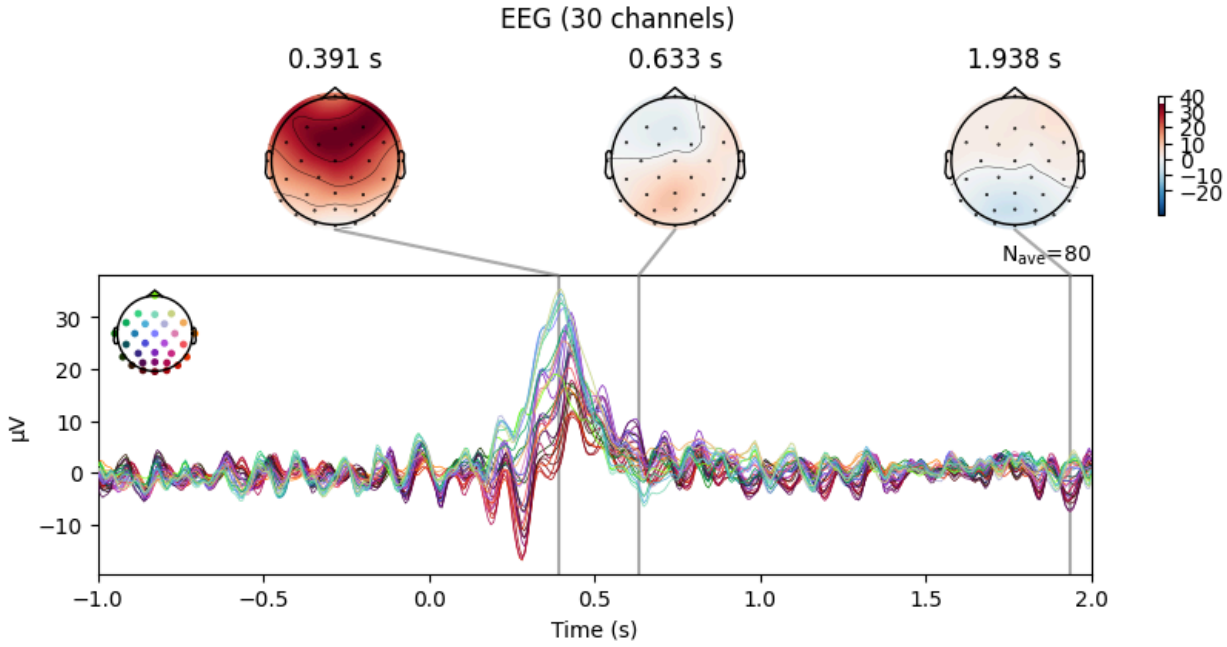
Out[38]:



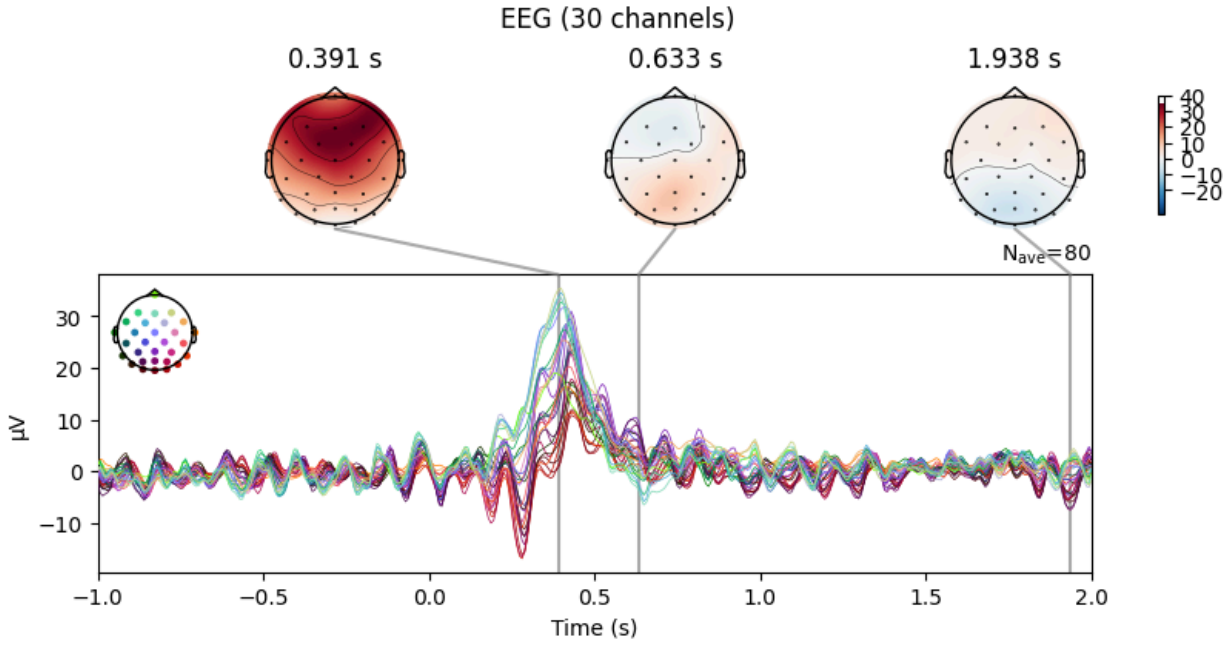
绘制联合图

In [39]:
```
evoked.plot_joint()
```

```
No projector specified for this dataset. Please consider the method self.add_proj.
```

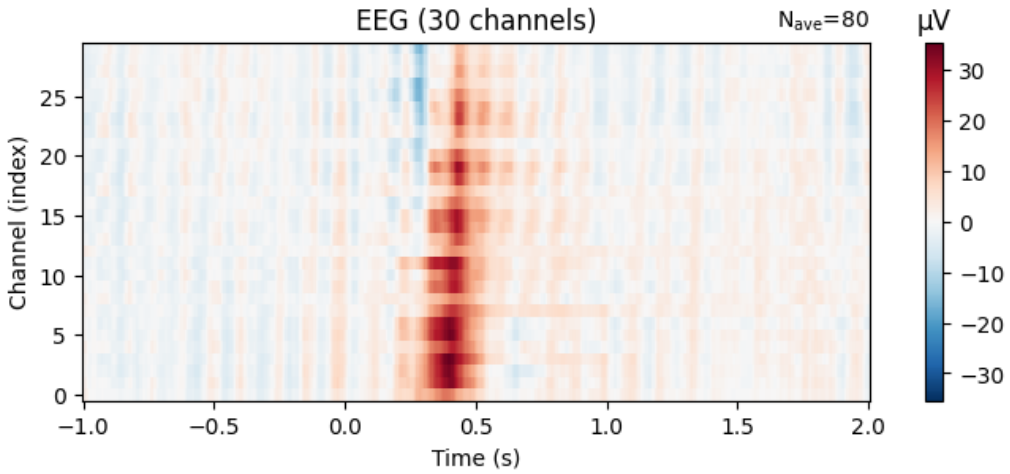


Out[39]:

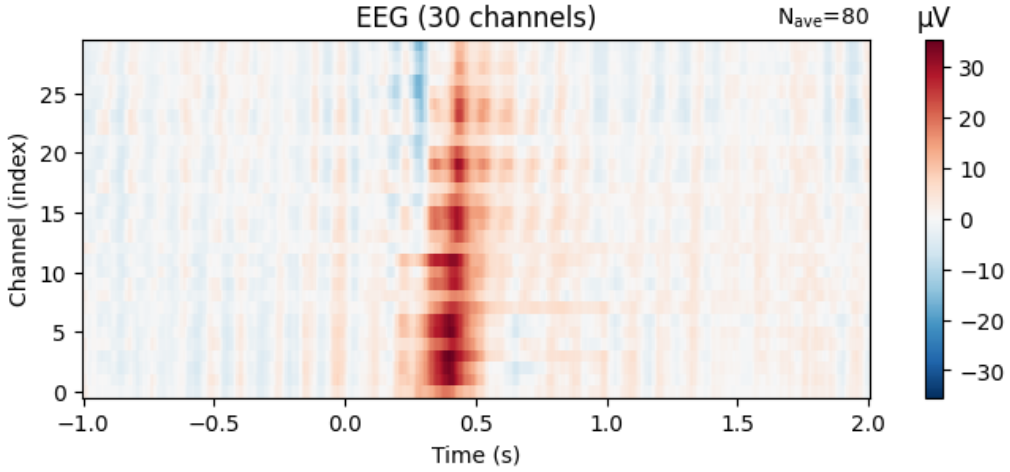


绘制逐导联热力图

In [40]:
```
evoked.plot_image()
```

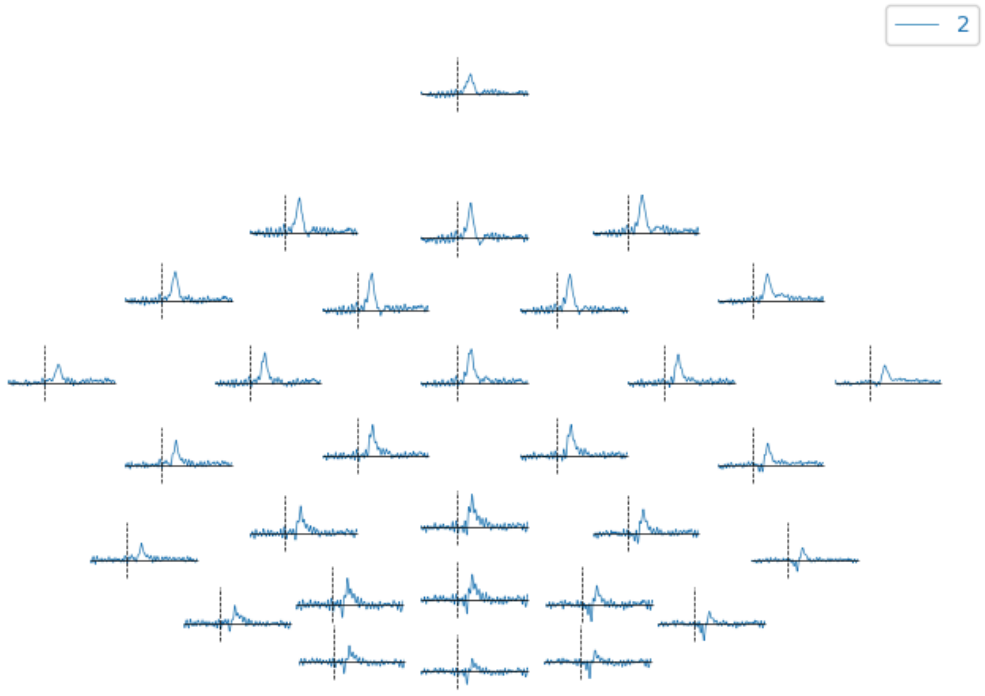


Out[40]:

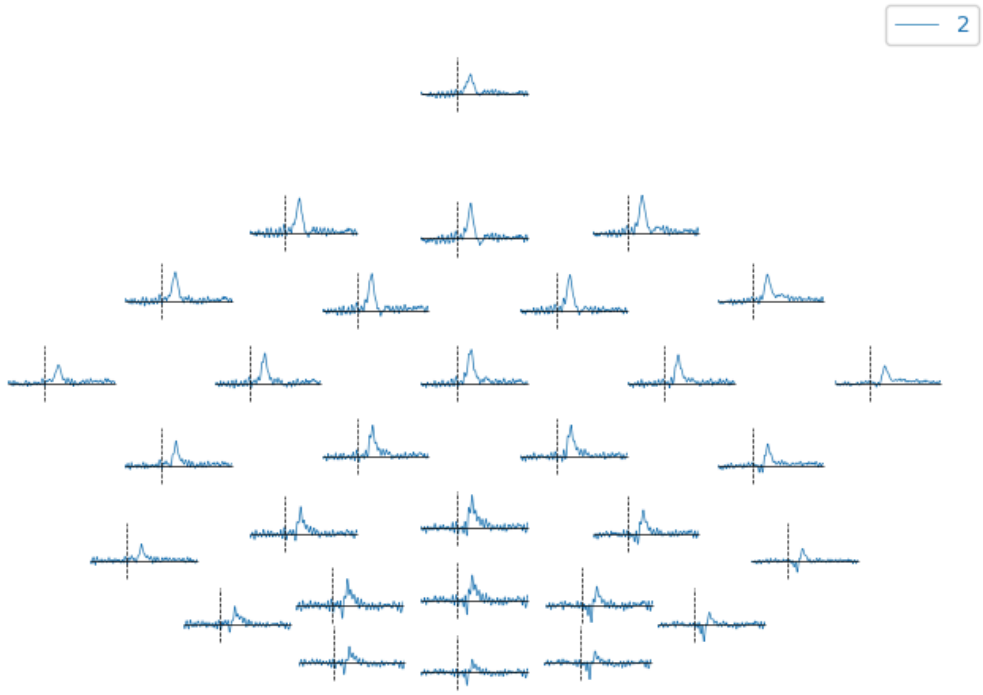


绘制拓扑时序信号图

In [41]:
```
evoked.plot_topo()
```



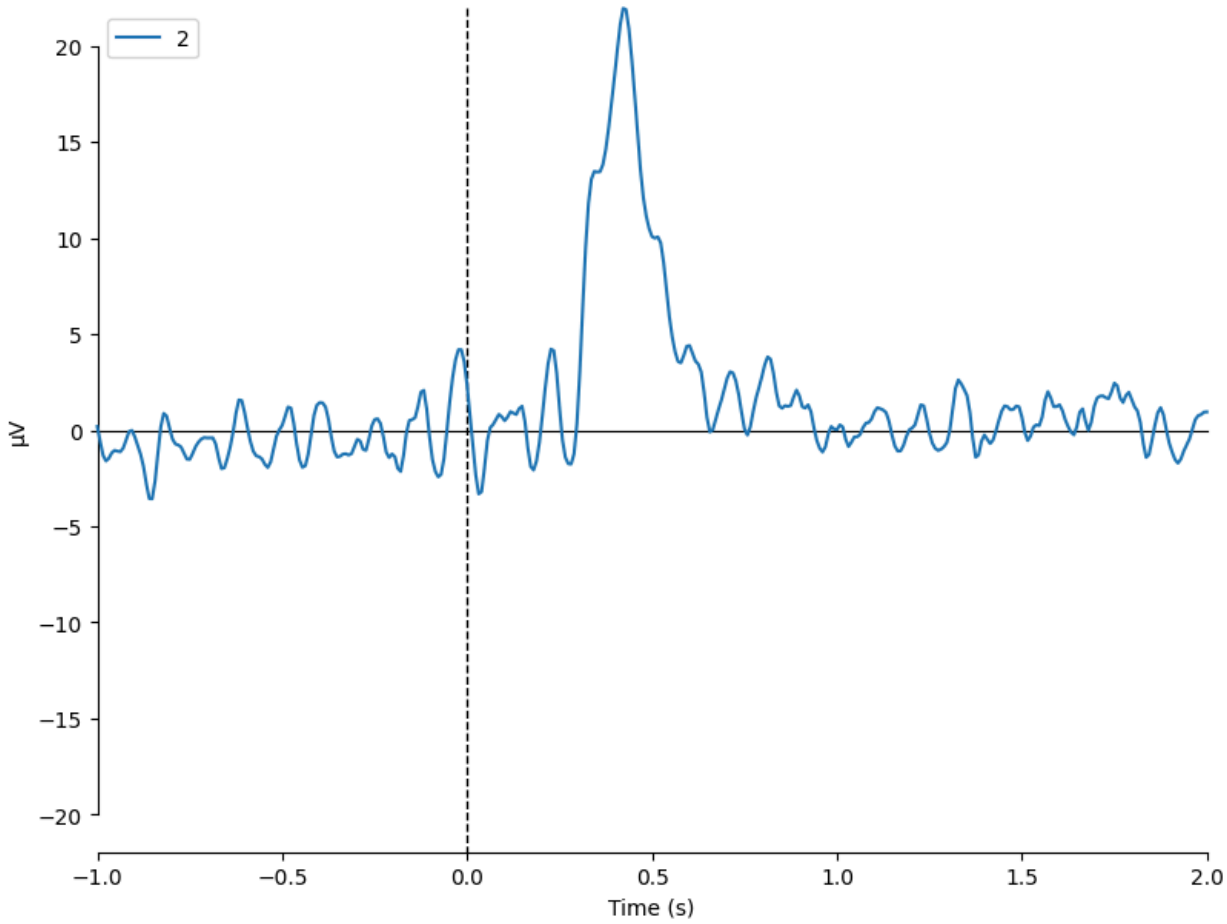
Out[41]:



绘制平均所有电极后的ERP

In [42]:
```
mne.viz.plot_compare_evokeds(evokeds=evoked, combine='mean')
```

```
combining channels using "mean"
```

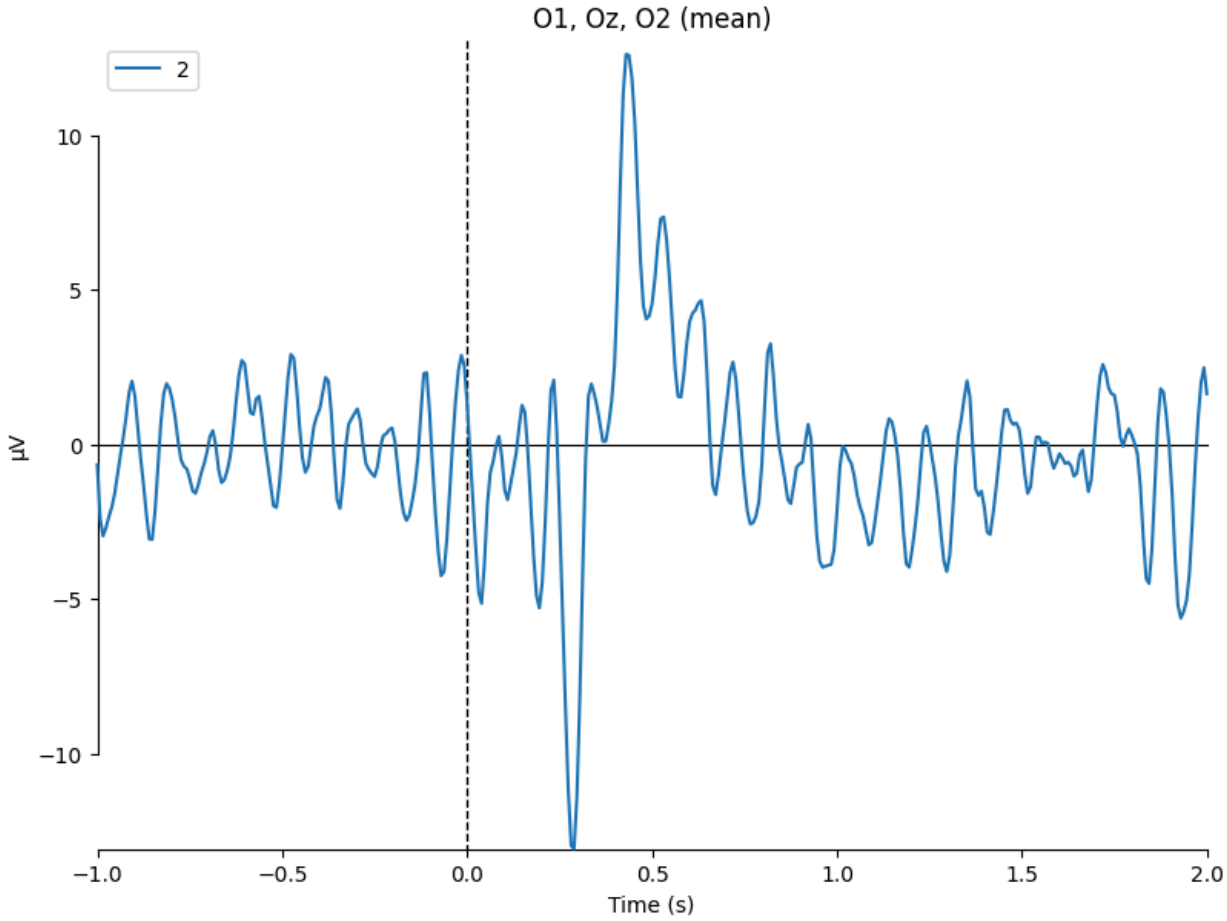


Out[42]:
```
[<Figure size 800x600 with 1 Axes>]
```

绘制枕叶电极的平均ERP

In [43]:
```
mne.viz.plot_compare_evokeds(evokeds=evoked, picks=['O1', 'Oz', 'O2'], combine='mean')
```

```
combining channels using "mean"
```



Out[43]: [<Figure size 800x600 with 1 Axes>]

第七步 - 时频分析

MNE提供了三种时频分析计算方法，分别是：

- Morlet wavelets，对应mne.time_frequency.tfr_morlet()
- DPSS tapers，对应mne.time_frequency.tfr_multitaper()
- Stockwell Transform，对应mne.time_frequency.tfr_stockwell()

这里，使用第一种方法为例

时频分析

计算能量（Power）与试次间耦合（inter-trial coherence，ITC）

In [44]:
```
# 设定一些时频分析的参数
# 频段选取4-30Hz

freqs = np.logspace(*np.log10([4, 30]), num=10)
n_cycles = freqs / 2.
power, itc = tfr_morlet(epochs, freqs=freqs, n_cycles=n_cycles, use_fft=True)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    1.8s finished
```

返回的power即为能量结果，itc即为试次间耦合结果
MNE中时频分析默认返回试次平均后的结果
如果想获取每个试次单独的时频分析结果，将average参数设为False即可
对应代码进行如下修改即可：

> power, itc = tfr_morlet(epochs, freqs=freqs, n_cycles=n_cycles, use_fft=True, average=False)

绘制时频结果

MNE的时频绘图方法里可以进行多种baseline矫正方法的选择
其对应参数为mode，包括以下一些选择：

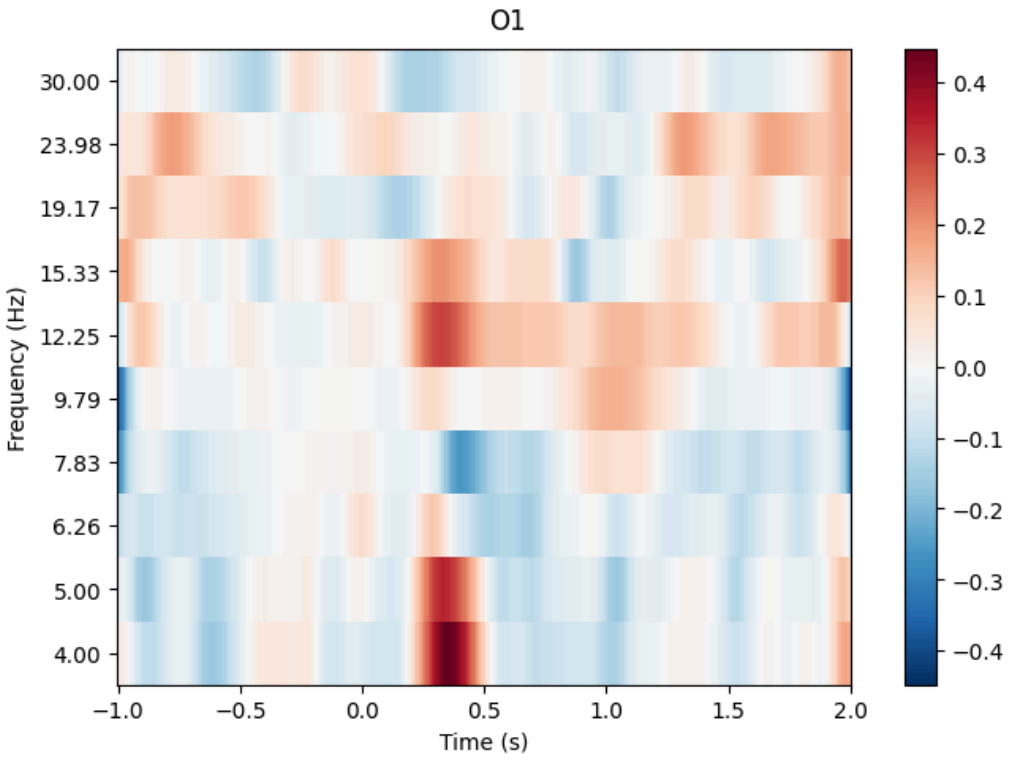
- 'mean'，减去baseline均值
- 'ratio'，除以baseline均值
- 'logratio'，除以baseline均值并取log
- 'percent'，减去baseline均值并除以baseline均值
- 'zscore'，减去baseline均值再除以baseline标准差
- 'zlogratio'，除以baseline均值并取log再除以baseline取log后的标准差

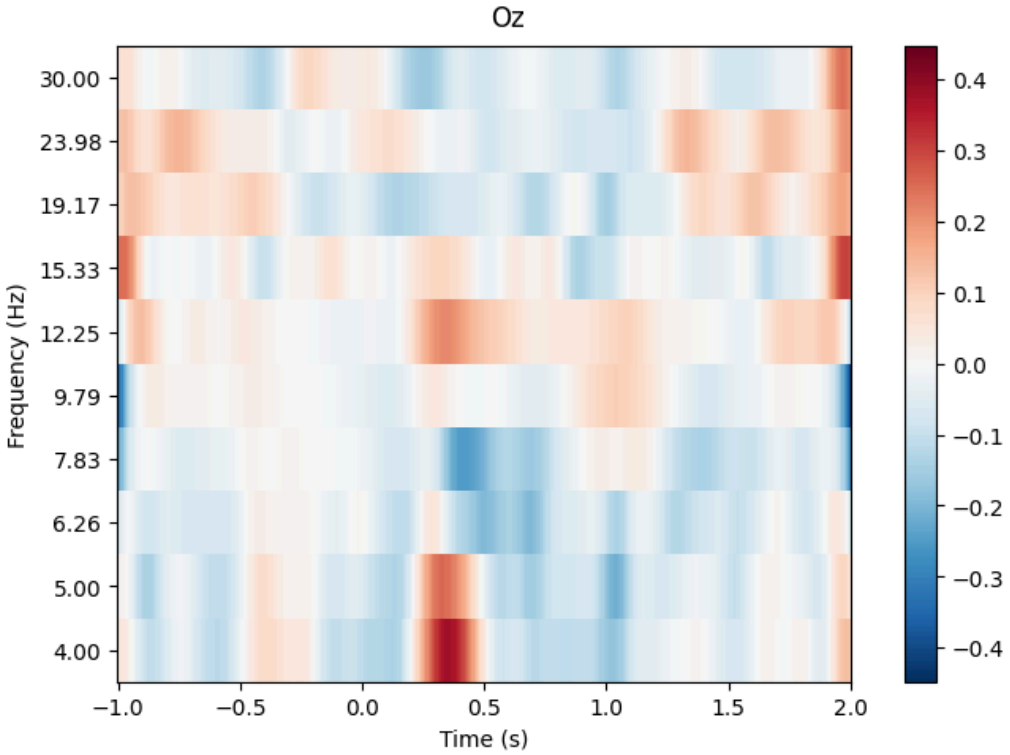
下例中选择logratio的方法进行基线校正

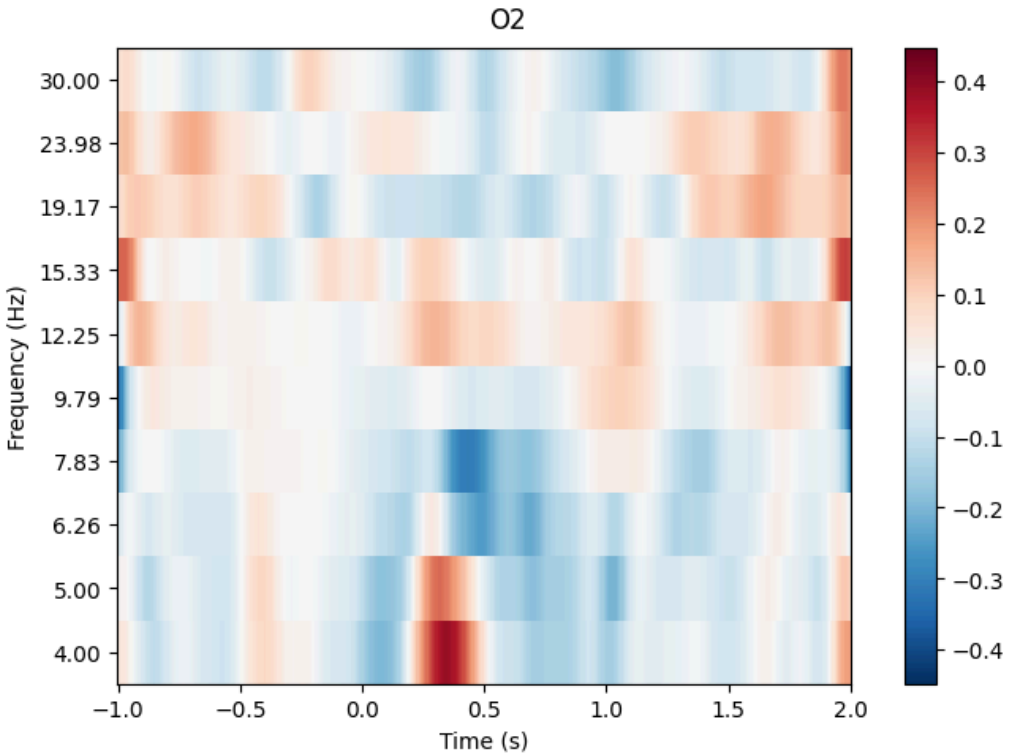
绘制枕叶导联的power结果

In [45]:
```
power.plot(picks=['O1', 'Oz', 'O2'], baseline=(-0.5, 0), mode='logratio', title='auto')
```

```
Applying baseline correction (mode: logratio)
```





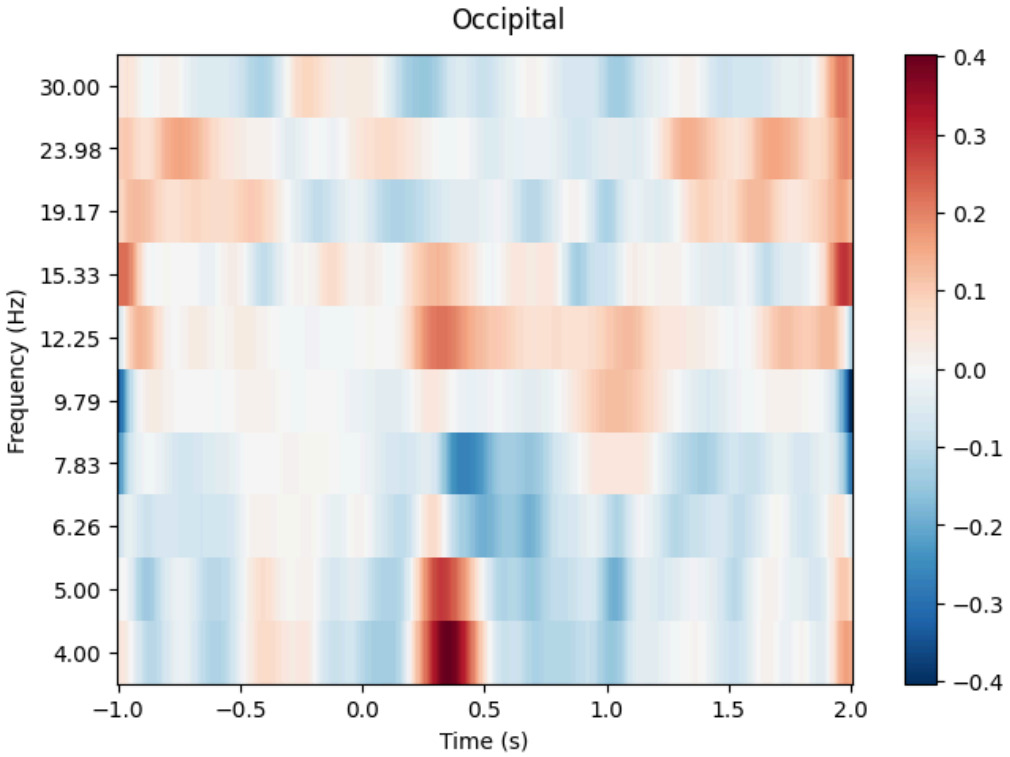


Out[45]:
```
[<Figure size 640x480 with 2 Axes>,
 <Figure size 640x480 with 2 Axes>,
 <Figure size 640x480 with 2 Axes>]
```

绘制枕叶导联的平均power结果

In [46]:
```
power.plot(picks=['O1', 'Oz', 'O2'], baseline=(-0.5, 0), mode='logratio',
           title='Occipital', combine='mean')
```

```
Applying baseline correction (mode: logratio)
```

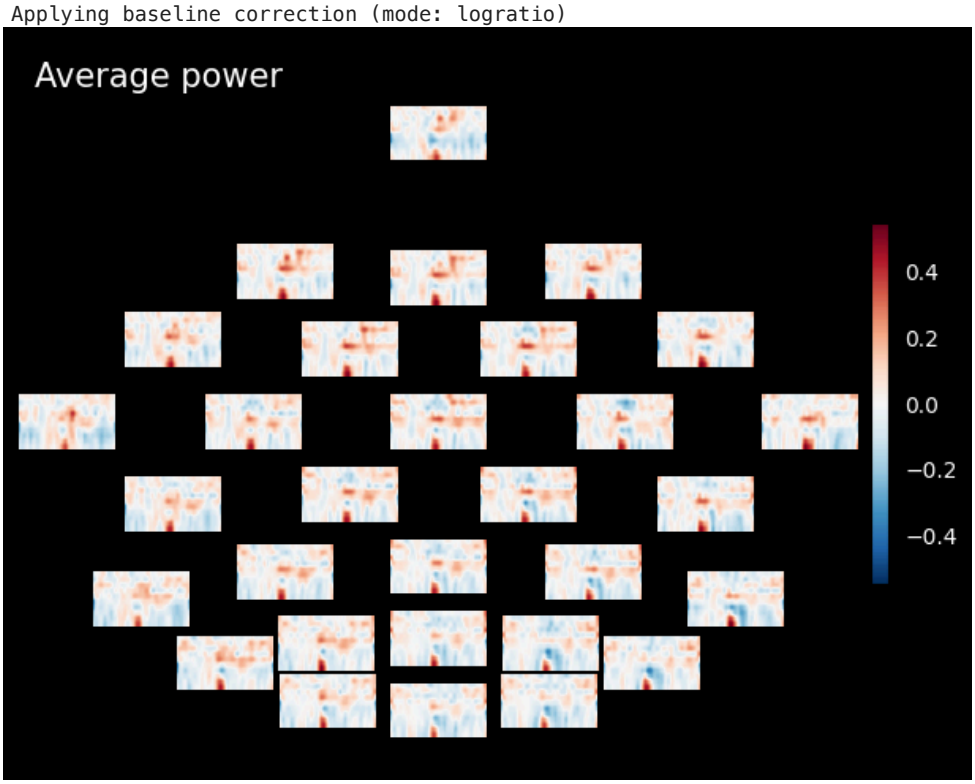


Out[46]: [<Figure size 640x480 with 2 Axes>]

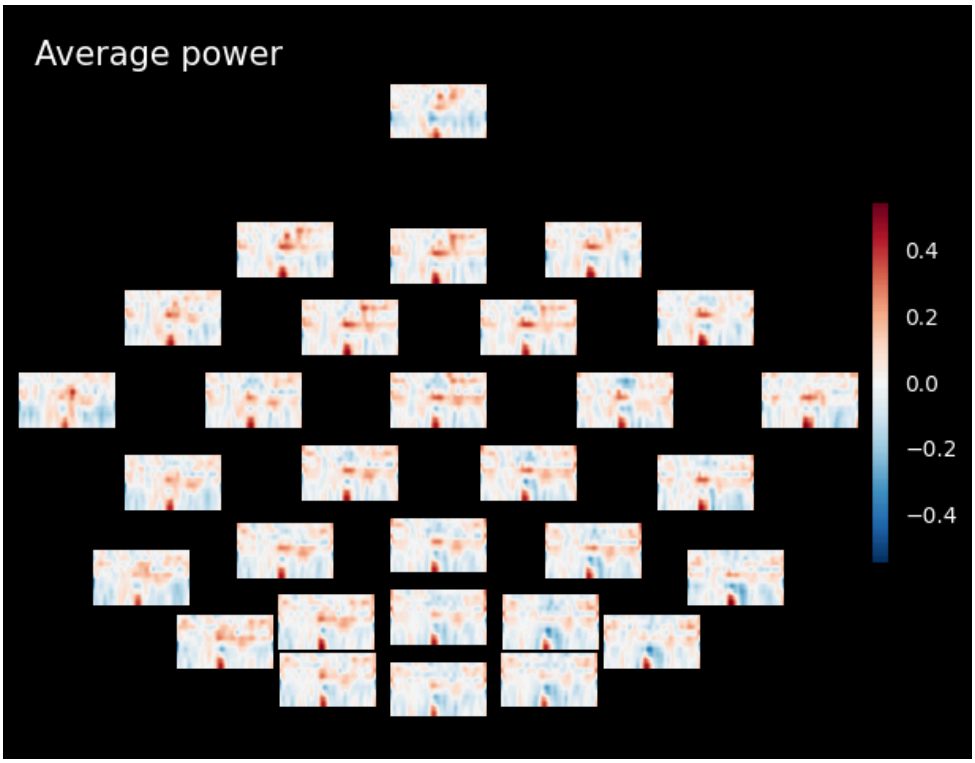
绘制power拓扑图

In [47]:
```
power.plot_topo(baseline=(-0.5, 0), mode='logratio', title='Average power')
```

```
Applying baseline correction (mode: logratio)
```



Out[47]:

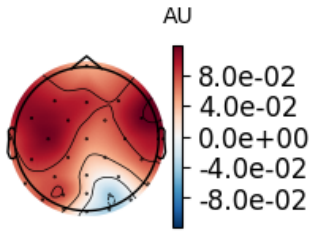


绘制不同频率的power拓扑图

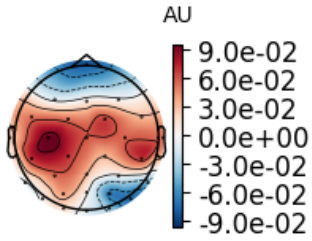
In [48]:

```
# 以theta power和alpha power为例
# 取0-0.5s的结果

power.plot_topomap(tmin=0, tmax=0.5, fmin=4, fmax=8,
                   baseline=(-0.5, 0), mode='logratio')
power.plot_topomap(tmin=0, tmax=0.5, fmin=8, fmax=12,
                   baseline=(-0.5, 0), mode='logratio')
```

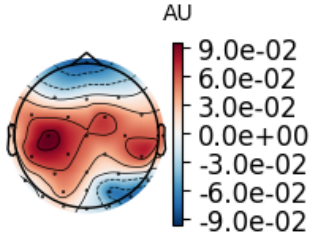
```
Applying baseline correction (mode: logratio)
```




```
Applying baseline correction (mode: logratio)
```



Out[48]:



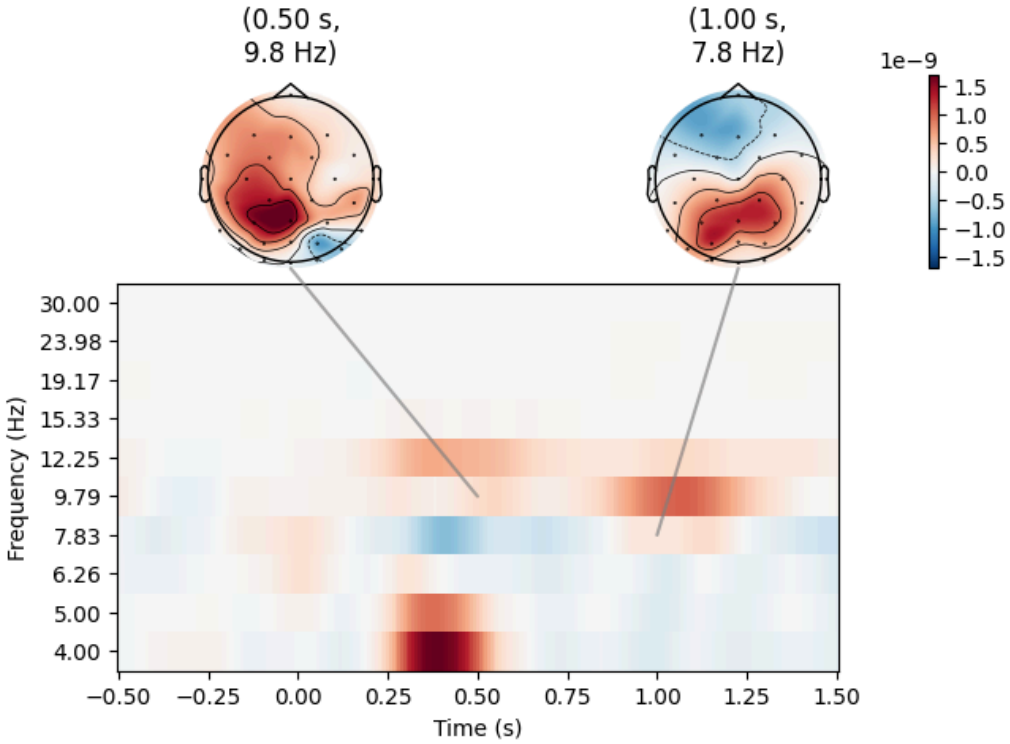
绘制联合图

In [49]:

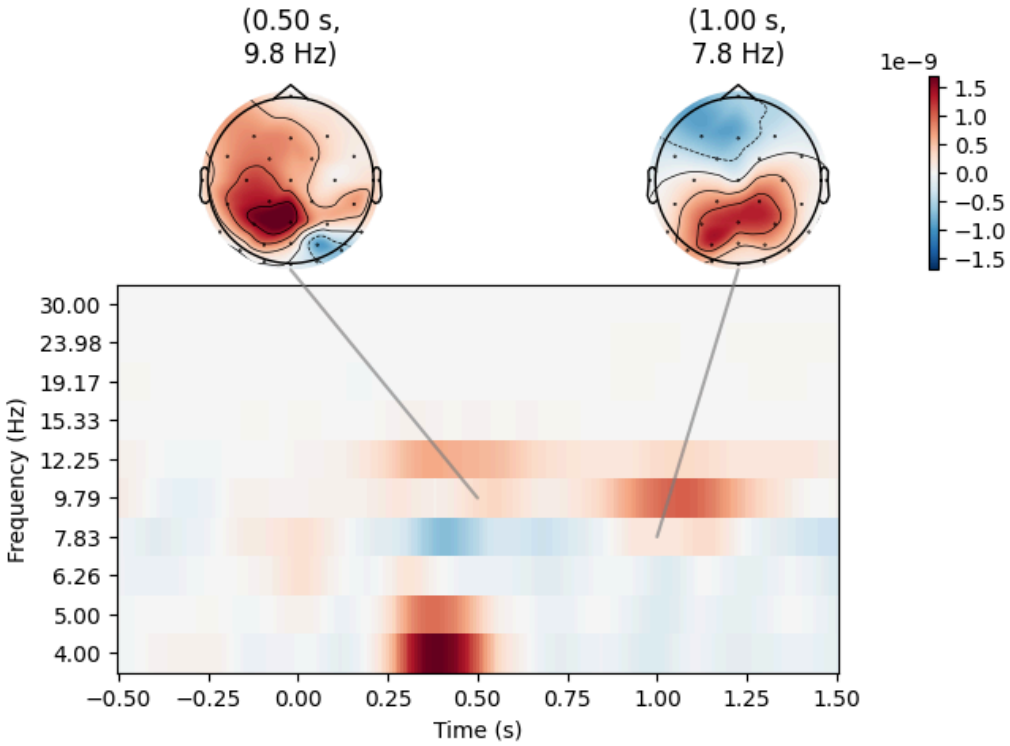
```
# 取-0.5s至1.5s的结果
# 并绘制0.5s时10Hz左右的结果和1s时8Hz左右的结果
```

```
power.plot_joint(baseline=(-0.5, 0), mode='mean', tmin=-0.5, tmax=1.5,
                 timefreqs=[(0.5, 10), (1, 8)])
```

```
Applying baseline correction (mode: mean)
Applying baseline correction (mode: mean)
Applying baseline correction (mode: mean)
```



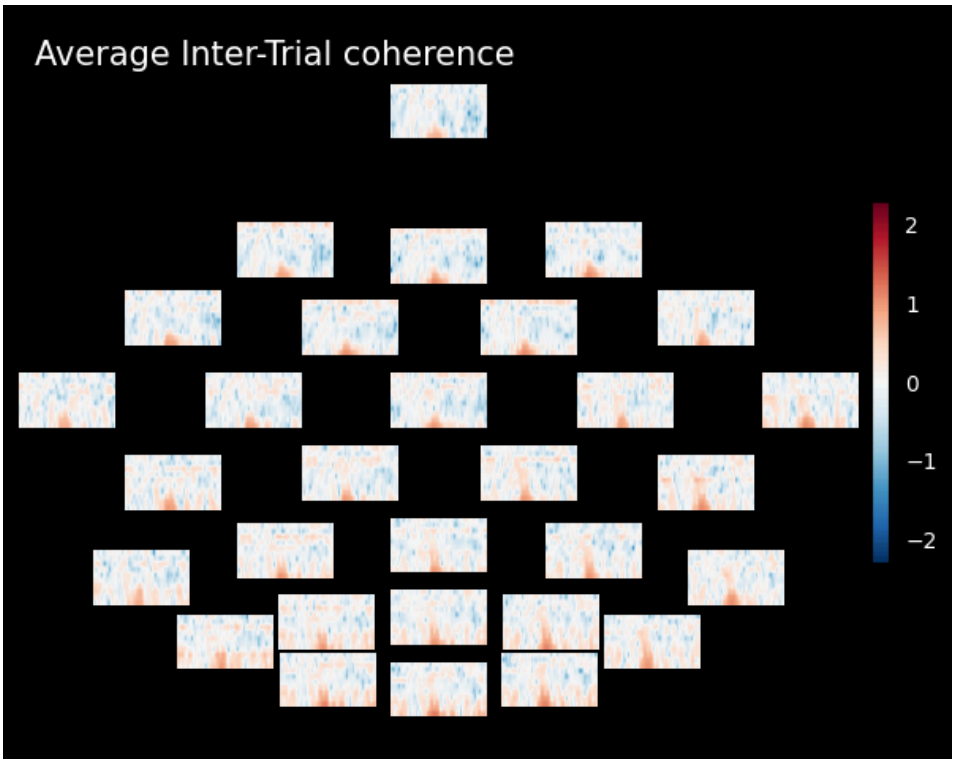
Out[49]:



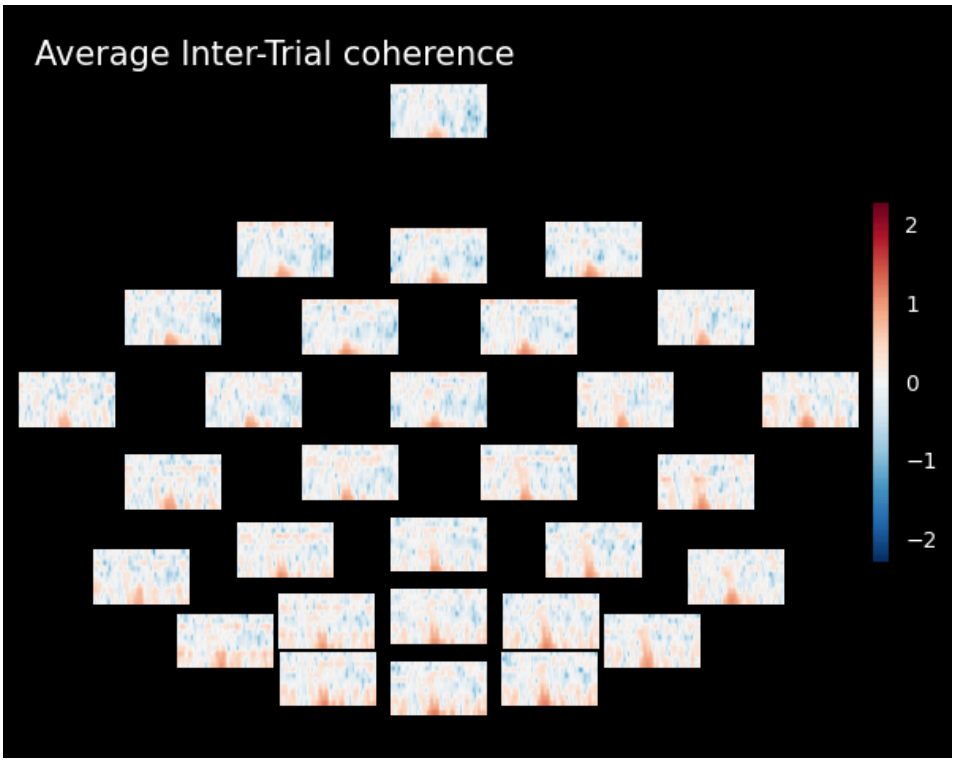
ITC结果的绘制是类似的，下面以拓扑图为例

In [50]:
```
itc.plot_topo(baseline=(-0.5, 0), mode='logratio', title='Average Inter-Trial coherence')
```

```
Applying baseline correction (mode: logratio)
```



Out[50]:



第八步 - 数据提取

在进行相关计算后，往往希望能提取原始数据矩阵、分段数据矩阵、时频结果矩阵等等
MNE中，Raw类（原始数据类型）、Epochs类（分段后数据类型）和Evocked类（叠加平均后数据类型）提供了get_data()方法
AverageTFR类（时频分析后数据类型）提供了.data属性

使用"get_data()"函数

以epochs为例：

In [51]:
```
epochs_array = epochs.get_data()
```

```
/var/folders/48/ln_7q6vj41sdf4mst103k9q80000gn/T/ipykernel_27170/4136209799.py:1: FutureWarning: The
current default of copy=False will change to copy=True in 1.7. Set the value of copy explicitly to a
void this warning
  epochs_array = epochs.get_data()
```

检查数据：

In [52]:
```
print(epochs_array.shape)
print(epochs_array)
```

```
(80, 32, 385)
[[[ 1.19955310e-05  2.64076199e-05  2.96414414e-05 ...  1.01356155e-05
    9.80684637e-06  5.92092751e-06]
  [ 1.90724252e-05  3.49982830e-05  1.52169891e-05 ... -6.16457047e-05
   -6.91169876e-05 -4.49072670e-05]
  [ 2.05216862e-05  3.59469862e-05  3.94453172e-05 ...  1.94767576e-05
    2.39333132e-05  2.49577850e-05]
  ...
  [-8.66725318e-07  8.10705304e-06  1.19969864e-05 ...  2.82622616e-05
    2.52288343e-05  2.71230860e-05]
  [-2.53263466e-06  7.27968446e-06  1.04266239e-05 ...  1.93294562e-05
    1.69034508e-05  2.34771912e-05]
  [ 1.33649215e-07  1.23793332e-05  1.50257433e-05 ...  1.05866334e-05
    9.54775567e-06  2.08057745e-05]]

 [[-2.86156092e-05 -2.46272912e-05 -2.49375585e-05 ... -5.66655490e-06
   -7.52341851e-06 -3.20816720e-06]
  [-1.98700747e-05 -2.32499213e-05 -2.34376177e-05 ... -5.87000498e-05
   -6.47002100e-05 -5.75247492e-05]
  [-4.77415730e-05 -4.28926395e-05 -4.62196702e-05 ... -1.70584942e-05
   -1.71572445e-05 -9.82390602e-06]
  ...
  [-1.60671591e-05 -1.70554320e-05 -1.78535513e-05 ... -2.06767937e-05
   -2.10371919e-05 -1.60745797e-05]
  [-1.27360726e-05 -1.29404692e-05 -1.56404187e-05 ... -2.57404057e-05
   -2.57716814e-05 -2.15169283e-05]
  [-3.91312740e-06 -4.33246287e-06 -8.74230064e-06 ... -2.92476907e-05
   -3.12189701e-05 -2.55777220e-05]]

 [[ 6.85227991e-06  1.32988569e-05  1.17931526e-05 ... -1.05257553e-05
   -7.29973247e-06  7.60811272e-07]
  [ 4.57868772e-05  6.10002561e-05  4.86857343e-05 ... -5.88723235e-05
   -4.53612780e-05 -5.21097750e-05]
  [-2.39041747e-06  5.51625605e-07 -3.43916043e-06 ... -2.09373615e-05
   -1.64900388e-05 -8.28851746e-06]
  ...
  [-1.33391776e-05 -1.07467635e-05 -1.24442829e-05 ... -1.64522834e-07
   -6.02052540e-06 -1.69078225e-05]
  [-2.08669303e-05 -1.78934662e-05 -1.64818362e-05 ...  3.76944599e-06
   -3.30115813e-06 -1.70946462e-05]
  [-2.83501975e-05 -2.21030099e-05 -1.98491421e-05 ...  8.76663948e-06
    1.64174717e-06 -1.52025402e-05]]

 ...

 [[ 6.80863318e-06  1.05524233e-05  6.72074726e-06 ... -4.01385062e-06
   -9.19784618e-07  1.06133759e-05]
  [ 5.76037713e-06  2.66259834e-05  8.75669214e-06 ... -3.11041534e-05
   -2.78780029e-05 -2.83221310e-05]
  [ 2.17025296e-05  3.54538639e-05  3.09487299e-05 ... -1.52174122e-05
   -1.63483974e-05  5.62132499e-06]
  ...
  [ 1.20074873e-05  2.12530233e-05  1.63303138e-05 ... -3.10716566e-06
    1.92047329e-06  6.47991816e-06]
  [ 1.46415609e-05  2.31887717e-05  1.69637313e-05 ...  7.89310003e-06
    1.46942781e-05  1.68083116e-05]
  [ 1.97671096e-05  2.89008509e-05  2.27972561e-05 ...  8.55728169e-06
    1.30367610e-05  1.41579469e-05]]

 [[ 1.74816850e-05  2.49819499e-05  2.29105866e-05 ...  3.81813476e-05
    4.09137267e-05  3.66925733e-05]
  [ 6.84401370e-06  8.58840419e-06  1.42037583e-06 ...  3.12988143e-05
    4.72432342e-05  3.72120719e-05]
  [ 2.43129604e-05  4.65274677e-05  4.61777299e-05 ...  2.78136110e-05
    3.18060014e-05  3.07669788e-05]
  ...
  [ 3.25606845e-06  8.07309023e-06  8.90675997e-06 ... -1.18110444e-05
   -4.80246809e-06 -9.39073249e-06]
  [ 1.25935768e-06  1.87061553e-06  1.89885310e-06 ... -1.64250945e-05
   -1.53387943e-05 -2.01224886e-05]
  [-3.15113541e-06 -5.09472319e-06 -6.65621545e-06 ... -2.17458072e-05
   -2.18670127e-05 -2.41018918e-05]]

 [[ 3.64681540e-05  2.98502815e-05  2.87498156e-05 ...  1.55175056e-05
    1.44826216e-05  1.30606681e-05]
  [ 2.55042060e-05  2.24582884e-05  1.90968657e-05 ...  2.20694343e-05
    1.64899586e-05  1.74567354e-05]
  [ 3.97881502e-05  3.42533154e-05  3.30624925e-05 ...  1.20407018e-05
    1.41817010e-05  2.08039910e-05]
  ...
  [-2.56909640e-05 -2.45978874e-05 -7.55036844e-06 ... -1.65130791e-05
```

```
  -1.38872442e-05 -7.89003310e-06]
 [-2.81632418e-05 -2.33694463e-05 -6.84977804e-06 ... -1.02391454e-05
  -1.11644715e-05 -9.53458652e-06]
 [-2.83165245e-05 -2.30232812e-05 -9.50182053e-06 ... -9.52254434e-06
  -1.06435252e-05 -1.02143019e-05]]]
```

即获取了NumPy Array形式的分段数据

其shape为[80, 32, 385]

分别对应80个试次，32个导联和385个时间点

若想获取eog外的导联数据，则可将上述代码改为：

> epochs_array = epochs.get_data(picks=['eeg'])

使用".data"方法

In [53]:
```
power_array = power.data
```

检查数据：

In [54]:
```
print(power_array.shape)
print(power_array)
```

```
(30, 10, 385)
[[[8.85986698e-10 9.07973706e-10 9.20058263e-10 ... 1.23420496e-09
   1.24995475e-09 1.24461341e-09]
  [7.32918844e-10 7.64367275e-10 7.87204341e-10 ... 9.84102647e-10
   9.74923498e-10 9.50824443e-10]
  [6.82156256e-10 7.29602613e-10 7.70687201e-10 ... 9.47208394e-10
   9.01253689e-10 8.46802067e-10]
  ...
  [8.68702339e-11 9.23412004e-11 9.71742416e-11 ... 1.04863686e-10
   9.93734589e-11 9.31959758e-11]
  [5.78475285e-11 6.18542974e-11 6.54692807e-11 ... 6.66801457e-11
   6.31275474e-11 5.91835037e-11]
  [3.59707583e-11 3.76646671e-11 3.89359327e-11 ... 4.17503064e-11
   4.02655814e-11 3.83566425e-11]]

 [[2.08549861e-09 2.15718563e-09 2.20604952e-09 ... 3.20895308e-09
   3.24354417e-09 3.22755190e-09]
  [1.67844510e-09 1.76829060e-09 1.84030892e-09 ... 2.54055386e-09
   2.51505487e-09 2.45304872e-09]
  [1.49721569e-09 1.61508215e-09 1.72157081e-09 ... 2.29404750e-09
   2.19507946e-09 2.07371169e-09]
  ...
  [1.95450368e-10 2.07028295e-10 2.17023523e-10 ... 2.28229601e-10
   2.18259949e-10 2.06502462e-10]
  [1.35133996e-10 1.44986198e-10 1.53958015e-10 ... 1.50674004e-10
   1.43372628e-10 1.35093542e-10]
  [8.47464664e-11 8.91013298e-11 9.25126929e-11 ... 9.65906831e-11
   9.31846381e-11 8.88114337e-11]]

 [[2.22295166e-09 2.31660017e-09 2.38604514e-09 ... 3.51045316e-09
   3.54134285e-09 3.51494742e-09]
  [1.77295781e-09 1.88450427e-09 1.97858384e-09 ... 2.74909783e-09
   2.71253358e-09 2.63627140e-09]
  [1.51076595e-09 1.63970135e-09 1.75859273e-09 ... 2.41453915e-09
   2.30404228e-09 2.17050331e-09]
  ...
  [1.96780351e-10 2.08829235e-10 2.19371880e-10 ... 2.34532464e-10
   2.23558232e-10 2.10880473e-10]
  [1.43061998e-10 1.53952322e-10 1.64030574e-10 ... 1.51420722e-10
   1.45466513e-10 1.38281328e-10]
  [8.19397181e-11 8.62056052e-11 8.96162121e-11 ... 1.08230898e-10
   1.04721206e-10 1.00021570e-10]]

 ...

 [[9.84753972e-10 1.01080219e-09 1.02145662e-09 ... 1.42596061e-09
   1.41214770e-09 1.37788149e-09]
  [7.87141108e-10 8.13893902e-10 8.27357786e-10 ... 1.15045298e-09
   1.11912311e-09 1.07333810e-09]
  [8.25047362e-10 8.78968894e-10 9.22526977e-10 ... 1.15141125e-09
   1.08797105e-09 1.01514380e-09]
  ...
  [1.42752146e-10 1.51785397e-10 1.59695581e-10 ... 1.83127076e-10
   1.76466908e-10 1.68068714e-10]
  [6.48164587e-11 6.74976999e-11 6.94854672e-11 ... 9.04567891e-11
   8.84664171e-11 8.55142632e-11]
  [4.57731069e-11 4.82475434e-11 5.02649826e-11 ... 7.27526735e-11
   7.01859193e-11 6.68705327e-11]]

 [[9.65946445e-10 9.93734977e-10 1.00735001e-09 ... 1.23711642e-09
   1.21981722e-09 1.18547472e-09]
  [7.93019581e-10 8.22586010e-10 8.39437238e-10 ... 9.94701252e-10
   9.65426266e-10 9.24016665e-10]
  [8.40246245e-10 8.95797556e-10 9.40779043e-10 ... 1.01838574e-09
   9.61625413e-10 8.96635599e-10]
  ...
  [1.28203840e-10 1.34680477e-10 1.39926354e-10 ... 1.62472689e-10
   1.57977015e-10 1.51762295e-10]
  [6.42709357e-11 6.65861212e-11 6.81186377e-11 ... 8.14601056e-11
   8.07555706e-11 7.89986809e-11]
  [4.31713285e-11 4.50276028e-11 4.63719334e-11 ... 6.97878090e-11
   6.77566077e-11 6.49078578e-11]]

 [[1.07297817e-09 1.10189041e-09 1.11549707e-09 ... 1.53222904e-09
   1.52549468e-09 1.49442694e-09]
  [8.73017183e-10 9.06061390e-10 9.25793251e-10 ... 1.18258153e-09
   1.16023297e-09 1.12059001e-09]
  [8.99004157e-10 9.58896787e-10 1.00808198e-09 ... 1.14042800e-09
   1.08719320e-09 1.02230790e-09]
  ...
  [1.25757828e-10 1.32188552e-10 1.37498259e-10 ... 1.61857580e-10
```

```
   1.58183566e-10 1.52745621e-10]
  [7.07738227e-11 7.37483707e-11 7.58665593e-11 ... 9.27531229e-11
   9.17143856e-11 8.94828847e-11]
  [4.88687480e-11 5.11797142e-11 5.29157649e-11 ... 7.73128267e-11
   7.50204552e-11 7.18113525e-11]]]
```

即获取了NumPy Array形式的时频power结果

其shape为[30, 10, 385]

分别对应30个导联，10个频率和385个时间点

第二章: 基础Python数据处理

根据在EEG数据处理过程中可能需要用到的一些基本数据处理技能，这一章旨在提供一个使用Python进行数组操作与统计分析的基本教程，其分为以下三个部分：

- **第一节: 基础NumPy数组操作**
- **第二节: 基础数据读取与存储操作**
- **第三节: 基础统计分析**

下载与导入Python包

In [1]:
```
!pip install h5py
!pip install neurora

import numpy as np
import h5py
import scipy.io as sio
import matplotlib.pyplot as plt
from scipy.stats import ttest_1samp, ttest_ind, ttest_rel, f_oneway
from mne.stats import fdr_correction, f_mway_rm
from neurora.stuff import clusterbased_permutation_1d_1samp_1sided, permutation_test
```

```
Requirement already satisfied: h5py in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (3.9.
0)
Requirement already satisfied: numpy>=1.17.3 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from h5py) (1.23.5)
Requirement already satisfied: neurora in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (1.
1.6.10)
Requirement already satisfied: numpy in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
neurora) (1.23.5)
Requirement already satisfied: scipy>=1.6.2 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from neurora) (1.11.1)
Requirement already satisfied: mne in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from n
eurora) (1.6.1)
Requirement already satisfied: nibabel in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (fr
om neurora) (5.2.0)
Requirement already satisfied: matplotlib in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from neurora) (3.6.3)
Requirement already satisfied: nilearn in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (fr
om neurora) (0.10.3)
Requirement already satisfied: scikit-learn in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from neurora) (1.3.0)
Requirement already satisfied: scikit-image in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from neurora) (0.20.0)
Requirement already satisfied: contourpy>=1.0.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from matplotlib->neurora) (1.0.5)
Requirement already satisfied: cycler>=0.10 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from matplotlib->neurora) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from matplotlib->neurora) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from matplotlib->neurora) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pack
ages (from matplotlib->neurora) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from matplotlib->neurora) (9.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from matplotlib->neurora) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /Users/zitonglu/anaconda3/lib/python3.11/site
-packages (from matplotlib->neurora) (2.8.2)
Requirement already satisfied: tqdm in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
mne->neurora) (4.65.0)
Requirement already satisfied: pooch>=1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne->neurora) (1.8.1)
Requirement already satisfied: decorator in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne->neurora) (5.1.1)
Requirement already satisfied: jinja2 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (fro
m mne->neurora) (3.1.2)
Requirement already satisfied: lazy-loader>=0.3 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from mne->neurora) (0.3)
Requirement already satisfied: joblib>=1.0.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from nilearn->neurora) (1.2.0)
Requirement already satisfied: lxml in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
nilearn->neurora) (4.9.3)
Requirement already satisfied: pandas>=1.1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from nilearn->neurora) (1.5.3)
Requirement already satisfied: requests>=2.25.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from nilearn->neurora) (2.31.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/zitonglu/anaconda3/lib/python3.11/site
-packages (from scikit-learn->neurora) (2.2.0)
Requirement already satisfied: networkx>=2.8 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from scikit-image->neurora) (3.1)
Requirement already satisfied: imageio>=2.4.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-packa
ges (from scikit-image->neurora) (2.31.1)
Requirement already satisfied: tifffile>=2019.7.26 in /Users/zitonglu/anaconda3/lib/python3.11/site-
packages (from scikit-image->neurora) (2023.4.12)
Requirement already satisfied: PyWavelets>=1.1.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from scikit-image->neurora) (1.4.1)
Requirement already satisfied: pytz>=2020.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from pandas>=1.1.5->nilearn->neurora) (2023.3.post1)
Requirement already satisfied: platformdirs>=2.5.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-
packages (from pooch>=1.5->mne->neurora) (3.10.0)
Requirement already satisfied: six>=1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (f
rom python-dateutil>=2.7->matplotlib->neurora) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/zitonglu/anaconda3/lib/python3.11/
site-packages (from requests>=2.25.0->nilearn->neurora) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from requests>=2.25.0->nilearn->neurora) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests>=2.25.0->nilearn->neurora) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests>=2.25.0->nilearn->neurora) (2023.7.22)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from jinja2->mne->neurora) (2.1.1)
```

第一节 - 基础NumPy数组操作

用Python进行数据处理时，NumPy矩阵往往是最可能用来进行分析操作的数据类型
在第一部分，我们介绍了可能在之后数据分析过程中会涉及到的一些基础而重要的NumPy矩阵操作及其实现

生成数组

In [2]:
```
# 生成shape为[3, 4]的全为0的矩阵
A = np.zeros([3, 4])
print(A.shape)
print(A)
```

```
(3, 4)
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [3]:
```
# 生成shape为[3, 4]的全为1的矩阵
A = np.ones([3, 4])
print(A.shape)
print(A)
```

```
(3, 4)
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

In [4]:
```
# 生成shape为[3, 4]的随机矩阵（随机值在0-1之间）
A = np.random.rand(3, 4)
print(A.shape)
print(A)
```

```
(3, 4)
[[0.23392924 0.22035835 0.5638377  0.48848769]
 [0.47104945 0.18138117 0.70215049 0.18883969]
 [0.43616516 0.63167613 0.50099703 0.24018979]]
```

In [5]:
```
# 生成shape为[3, 4]的随机矩阵（随机值在20-80之间）
A = np.random.uniform(low=20, high=80, size=[3, 4])
print(A.shape)
print(A)
```

```
(3, 4)
[[55.9454466  63.76207084 35.20959362 69.44105043]
 [47.23437158 75.80405389 74.18094595 34.43744156]
 [35.05809095 50.07752316 54.6770267  60.10960428]]
```

矩阵铺平为向量

In [6]:
```
A = np.ravel(A)
print(A.shape)
print(A)
```

```
(12,)
[55.9454466  63.76207084 35.20959362 69.44105043 47.23437158 75.80405389
 74.18094595 34.43744156 35.05809095 50.07752316 54.6770267  60.10960428]
```

修改数组尺寸

In [7]:
```
# 将A的shape修改为[3, 4]
A = np.reshape(A, (3, 4))
print(A.shape)
print(A)
```

```
(3, 4)
[[55.9454466  63.76207084 35.20959362 69.44105043]
 [47.23437158 75.80405389 74.18094595 34.43744156]
 [35.05809095 50.07752316 54.6770267  60.10960428]]
```

数组转置

转置二维矩阵

In [8]:
```
A = A.T
print(A.shape)
print(A)
```

```
(4, 3)
[[55.9454466  47.23437158 35.05809095]
 [63.76207084 75.80405389 50.07752316]
 [35.20959362 74.18094595 54.6770267 ]
 [69.44105043 34.43744156 60.10960428]]
```

转置多维矩阵

In [9]:
```
# 首先，生成一个shape为[2, 3, 4]的三维数组
B = np.random.rand(2, 3, 4)
print(B.shape)
print(B)
```

```
(2, 3, 4)
[[[0.28237637 0.77526207 0.13826887 0.90446258]
  [0.83693299 0.40884927 0.86007176 0.19918559]
  [0.56159299 0.44729714 0.45152516 0.81938012]]

 [[0.28728874 0.49376632 0.82602795 0.809945  ]
  [0.81126796 0.66065919 0.36565283 0.67277449]
  [0.0574314  0.05197957 0.12344968 0.58150866]]]
```

In [10]:
```
# 将A转置为shape为[2, 4, 3]的数组
B = np.transpose(B, (0, 2, 1))
print(B.shape)
print(B)
```

```
(2, 4, 3)
[[[0.28237637 0.83693299 0.56159299]
  [0.77526207 0.40884927 0.44729714]
  [0.13826887 0.86007176 0.45152516]
  [0.90446258 0.19918559 0.81938012]]

 [[0.28728874 0.81126796 0.0574314 ]
  [0.49376632 0.66065919 0.05197957]
  [0.82602795 0.36565283 0.12344968]
  [0.809945   0.67277449 0.58150866]]]
```

In [11]:
```
# 这里是另外一个例子:
# 将shape为[2, 4, 3]的数组转置成shape为[4, 3, 2]的数组
B = np.transpose(B, (1, 2, 0))
print(B.shape)
print(B)
```

```
(4, 3, 2)
[[[0.28237637 0.28728874]
  [0.83693299 0.81126796]
  [0.56159299 0.0574314 ]]

 [[0.77526207 0.49376632]
  [0.40884927 0.66065919]
  [0.44729714 0.05197957]]

 [[0.13826887 0.82602795]
  [0.86007176 0.36565283]
  [0.45152516 0.12344968]]

 [[0.90446258 0.809945  ]
  [0.19918559 0.67277449]
  [0.81938012 0.58150866]]]
```

数组合并

In [12]:
```
# 首先，生成一个shape为[4, 3, 3]的数组
C = np.random.rand(4, 3, 3)
print(C.shape)
print(C)
```

```
(4, 3, 3)
[[[0.08809861 0.95260854 0.1491686 ]
  [0.12986342 0.03184737 0.27373474]
  [0.05778086 0.18012803 0.34049261]]

 [[0.46405186 0.13893962 0.75624289]
  [0.71479078 0.49171166 0.21432559]
  [0.03929975 0.48174964 0.9563534 ]]

 [[0.88337122 0.66038061 0.47654903]
  [0.42979245 0.81191699 0.56522443]
  [0.4229371  0.4152601  0.64921869]]

 [[0.68095355 0.4697602  0.75024432]
  [0.07283738 0.94909662 0.0476452 ]
  [0.86601746 0.94838068 0.62035814]]]
```

In [13]:
```
# 将数组B（shape为[4, 3, 2]）与数组C（shape为[4, 3, 3]）合并
# 即对第三个维度进行合并，合并后数组shape为[4, 3, 5]
D = np.concatenate((B, C), axis=2)
print(D.shape)
print(D)
```

```
(4, 3, 5)
[[[0.28237637 0.28728874 0.08809861 0.95260854 0.1491686 ]
  [0.83693299 0.81126796 0.12986342 0.03184737 0.27373474]
  [0.56159299 0.0574314  0.05778086 0.18012803 0.34049261]]

 [[0.77526207 0.49376632 0.46405186 0.13893962 0.75624289]
  [0.40884927 0.66065919 0.71479078 0.49171166 0.21432559]
  [0.44729714 0.05197957 0.03929975 0.48174964 0.9563534 ]]

 [[0.13826887 0.82602795 0.88337122 0.66038061 0.47654903]
  [0.86007176 0.36565283 0.42979245 0.81191699 0.56522443]
  [0.45152516 0.12344968 0.4229371  0.4152601  0.64921869]]

 [[0.90446258 0.809945   0.68095355 0.4697602  0.75024432]
  [0.19918559 0.67277449 0.07283738 0.94909662 0.0476452 ]
  [0.81938012 0.58150866 0.86601746 0.94838068 0.62035814]]]
```

In [14]:
```
# 这里是另外一个例子：
# 生成一个shape为[1, 3, 2]的数组E
# 合并数组B和数组E，得到shape为[5, 3, 2]的数组F
E = np.random.rand(1, 3, 2)
F = np.concatenate((B, E), axis=0)
print(F.shape)
print(F)
```

```
(5, 3, 2)
[[[0.28237637 0.28728874]
  [0.83693299 0.81126796]
  [0.56159299 0.0574314 ]]

 [[0.77526207 0.49376632]
  [0.40884927 0.66065919]
  [0.44729714 0.05197957]]

 [[0.13826887 0.82602795]
  [0.86007176 0.36565283]
  [0.45152516 0.12344968]]

 [[0.90446258 0.809945  ]
  [0.19918559 0.67277449]
  [0.81938012 0.58150866]]

 [[0.99912164 0.39421159]
  [0.03843999 0.48001635]
  [0.54657219 0.11174902]]]
```

平均数组中的值

In [15]:
```
# 对数组B（shape为[4, 3, 2]）的第二个维度的值进行平均
B_mean = np.average(B, axis=1)
print(B_mean.shape)
print(B_mean)
```

```
(4, 2)
[[0.56030078 0.38532937]
 [0.54380283 0.40213503]
 [0.4832886  0.43837682]
 [0.64100943 0.68807605]]
```

将非NumPy对象转换成NumPy数组

In [16]:
```
G = [[1, 2], [3, 4]]
G_narry = np.array(G)
print(type(G))
print(type(G_narry))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

第二节 - 基础数据读取与存储操作

这部分，会介绍一些基本的基于Python用于数据（数组）读取与存储操作

基于MNE读取数据

参考“第一章：单被试数据预处理”中的”第一步 - 数据读取”和“第八步 - 数据提取”部分

使用h5py存储与读取数据

h5py库可以将数据存储为一种压缩性能很强的HDF5格式文件
HDF5文件与MATLAB中的mat文件类似
都是通过Key和dataset构成
Key可以理解成数据的名称，dataset可以理解具体的数据
但是HDF5具有比mat更强的压缩性能
HDF5文件的后缀为.h5

使用h5py存储数据

In [17]:
```
# 生成一个数据，其shape为[4, 5]
testdata = np.random.rand(4, 5)
# 调用h5py，将数据存为一个名为'test_data.h5'文件
f = h5py.File('test_data.h5', 'w')
# 使用Key+dataset的方式存储上述testdata矩阵，这里Key命名为'data'
f.create_dataset('data', data=testdata)
# 关闭调用
f.close()
```

使用h5py读取数据

In [18]:
```
# 读取数据
testdata = np.array(h5py.File('test_data.h5', 'r')['data'])
# 打印数据信息
print(testdata.shape)
print(testdata)
```

```
(4, 5)
[[0.3978598  0.8996023  0.54708416 0.73816992 0.02574653]
 [0.44200296 0.12362103 0.17315831 0.95591195 0.12356735]
 [0.52541798 0.48842702 0.8370089  0.00473001 0.67011308]
 [0.19024479 0.21131253 0.50230262 0.86971355 0.08896379]]
```

使用NumPy存储与读取数据

NumPy自带的savetxt()可以将二维矩阵存为.txt文件
其自带的loadtxt()则可以读取.txt文件中的数据

使用NumPy存储数据

In [19]:
```
# 将NumPy Array格式的testdata存成一个名为'test_data.npy'的文件
np.save('test_data.npy', testdata)
```

使用NumPy读取.npy文件

In [20]:
```
# 读取'test_data.npy'
testdata = np.load('test_data.npy')
# 打印数据信息
```

```
print(testdata.shape)
print(testdata)
```

```
(4, 5)
[[0.3978598  0.8996023  0.54708416 0.73816992 0.02574653]
 [0.44200296 0.12362103 0.17315831 0.95591195 0.12356735]
 [0.52541798 0.48842702 0.8370089  0.00473001 0.67011308]
 [0.19024479 0.21131253 0.50230262 0.86971355 0.08896379]]
```

使用NumPy存储二维数组为.txt文件与读取

NumPy自带的savetxt()可以将二维矩阵存为.txt文件
其自带的loadtxt()则可以读取.txt文件中的数据

使用NumPy存储二维数组形式的数据

In [21]:
```
# 将NumPy Array格式的testdata存成一个名为'test_data.txt'的文件
np.savetxt('test_data.txt', testdata)
```

使用NumPy读取.txt文件存储的二维数组数据

In [22]:
```
# 读取'test_data.txt'
testdata = np.loadtxt('test_data.txt')
# 打印数据信息
print(testdata.shape)
print(testdata)
```

```
(4, 5)
[[0.3978598  0.8996023  0.54708416 0.73816992 0.02574653]
 [0.44200296 0.12362103 0.17315831 0.95591195 0.12356735]
 [0.52541798 0.48842702 0.8370089  0.00473001 0.67011308]
 [0.19024479 0.21131253 0.50230262 0.86971355 0.08896379]]
```

由于上述基于NumPy的方法仅适用于二维数据
若实在想存储多维数据，可以先reshape为二维数据再进行存储
以下代码为一个示例：

In [23]:
```
# 生成一个shape为[2, 3, 4]的三维数组
testdata_3d = np.random.rand(2, 3, 4)
# reshape成[2, 12]
testdata_3d_to_2d = np.reshape(testdata_3d, (2, 12))
# 存为.txt文件
np.savetxt('test_data_3d_to_2d.txt', testdata_3d_to_2d)
# 读取数据
testdata_2d = np.loadtxt('test_data_3d_to_2d.txt')
# reshape成[2, 3, 4]
testdata_2d_to_3d = np.reshape(testdata_2d, (2, 3, 4))
```

将.mat文件读取为NumPy数组

Python中读取MATLAB下的.mat格式文件，主要通过以下两种方式：

使用scipy.io读取

以下代码仅为示例，无实际数据
filename为待读入.mat文件的地址，Key为待读入数据的Key

```
import scipy.io as sio
data = np.array(sio.loadmat(filename)[Key])
```

Reading with h5py

h5py也可以用来读取.mat文件，与读取.h5文件的步骤相同
上述已经介绍过，以下代码仅为示例，无实际数据
这一方法不支持MATLAB版本低于7.3的.mat文件

```
data = np.array(h5py.File(filename, 'r')[Key])
```

第三节 - 基础统计分析

在这一节中，我们会介绍一系列需要在进行EEG数据多被试组分析时使用的基础统计操作

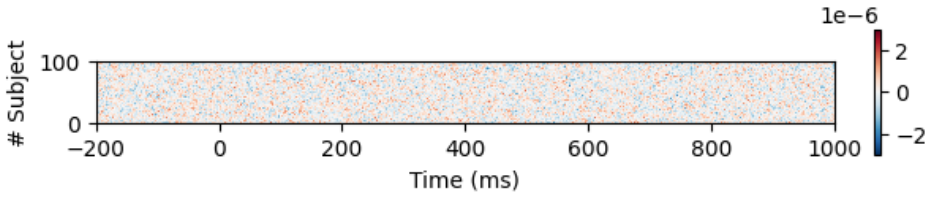
描述性统计

首先，我们介绍以下几个常用描述统计指标的计算方法:

- 均值(mean)
- 方差(variance)
- 标准差(standard deviation)
- 标准误差(standard error)

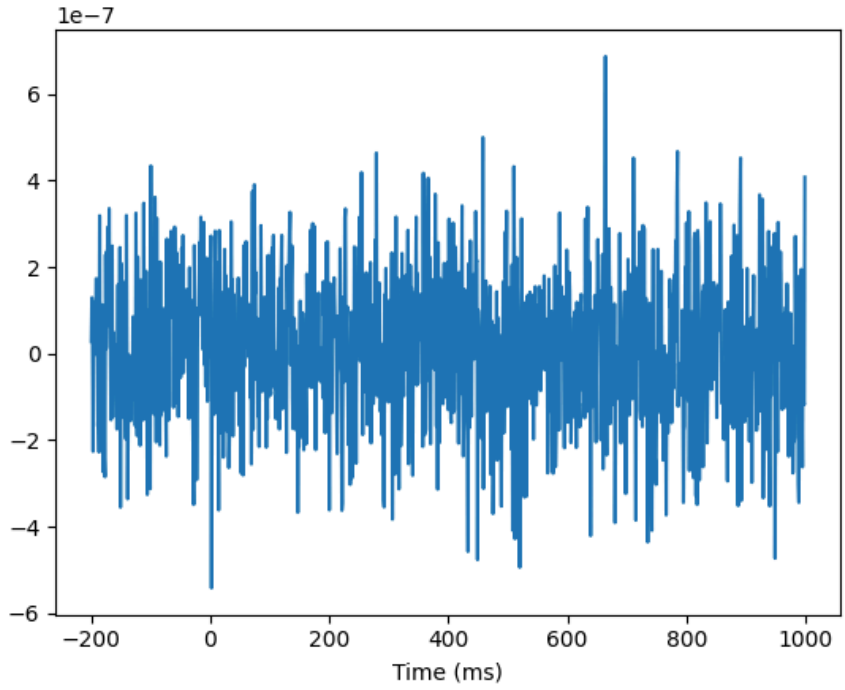
这里，生成一些虚假的脑电数据进行后续统计分析示例
假设随机生成如下的100个被试的从-200ms到1000ms（1200个时间点，每个点对应1ms）的事件相关电位（ERP）数据

In [24]:
```
# 生成虚假脑电数据，其shape为[100, 1200]
data = np.random.uniform(low=-3e-6, high=3e-6, size=(100, 1200))
# 逐trial可视化
plt.imshow(data, extent=[-200, 1000, 0, 100], cmap='RdBu_r')
plt.colorbar(fraction=0.008, ticks=[-2e-6, 0, 2e-6])
plt.xlabel('Time (ms)')
plt.ylabel('# Subject')
plt.show()
```



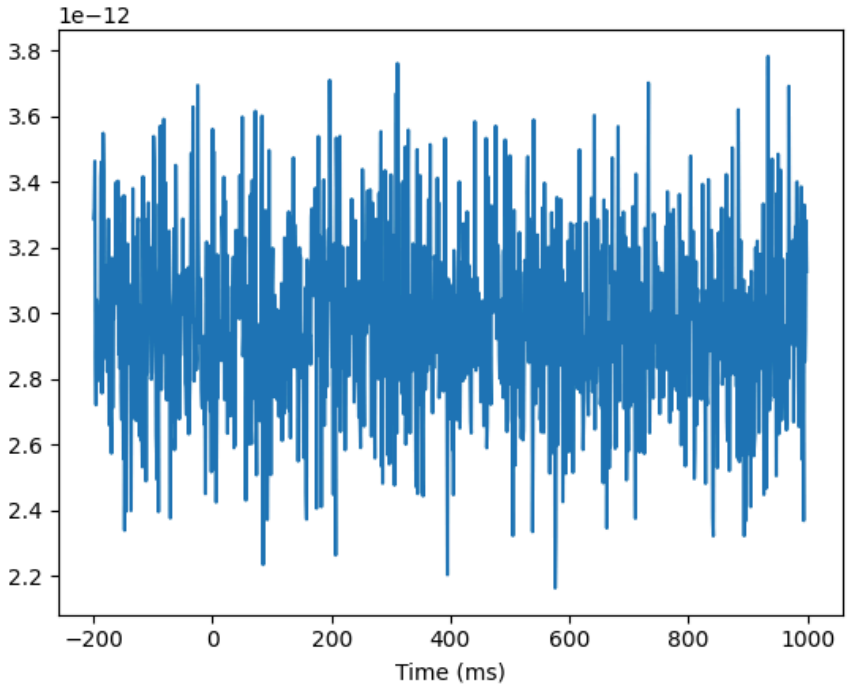
计算100个被试的ERP均值

In [25]:
```
# 对100个被试平均
data_mean = np.mean(data, axis=0)
times = np.arange(-200, 1000)
# 对均值可视化
plt.plot(times, data_mean)
plt.xlabel('Time (ms)')
plt.show()
```



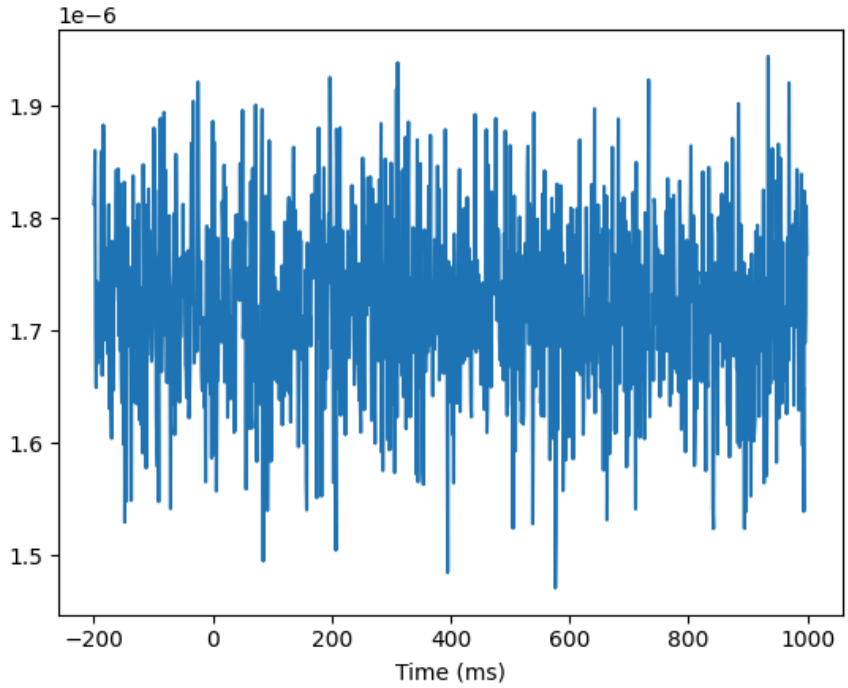
计算100个被试的ERP的方差

In [26]:
```
data_var = np.var(data, axis=0)
plt.plot(times, data_var)
plt.xlabel('Time (ms)')
plt.show()
```



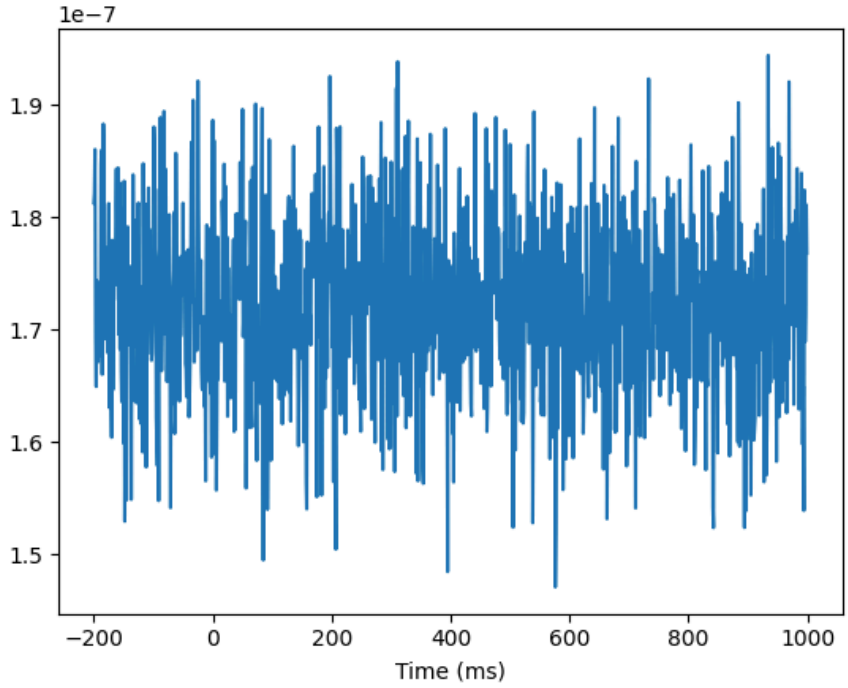
计算100个被试的ERP的标准差

In [27]:
```
data_std = np.std(data, axis=0)
plt.plot(times, data_std)
plt.xlabel('Time (ms)')
plt.show()
```



计算100个被试的ERP的标准误

In [28]:
```
n_subjects = 100
data_sem = np.std(data, axis=0, ddof=0)/np.sqrt(n_subjects)
plt.plot(times, data_sem)
plt.xlabel('Time (ms)')
plt.show()
```



推断性统计

在EEG研究中，除了对脑电数据的均值和离散程度进行计算外，还经常涉及到对总体特征的一些推断
对一些基本概念不再赘述，下面仅对常用的显著性检验方法进行介绍

单群组检验

当需要检验某一条件下的数据是否显著与某个特定的值不同，
常使用单样本t检验或置换检验(permutation_test)
这里，类似前面的例子，我们再次随机生成20个被试的1200个时间点的数据进行后续示例

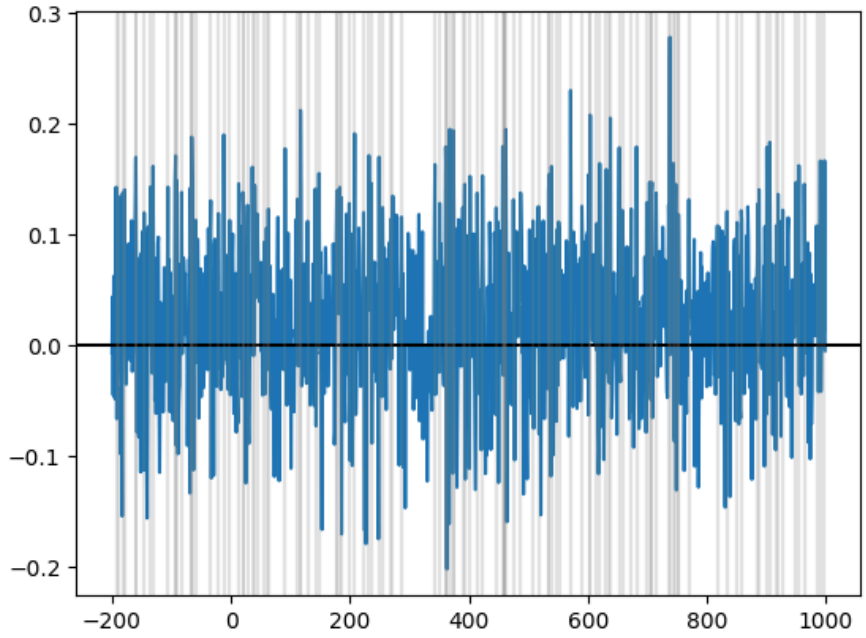
In [29]:
```
# 生成随机范围在-0.1到0.4之间的shape为[20, 1200]的数据
data = np.random.uniform(low=-0.5, high=0.55, size=(20, 1200))
```

参数检验方法：单样本t检验（未矫正）

进行逐时间点假设数据的值大于0的统计检验

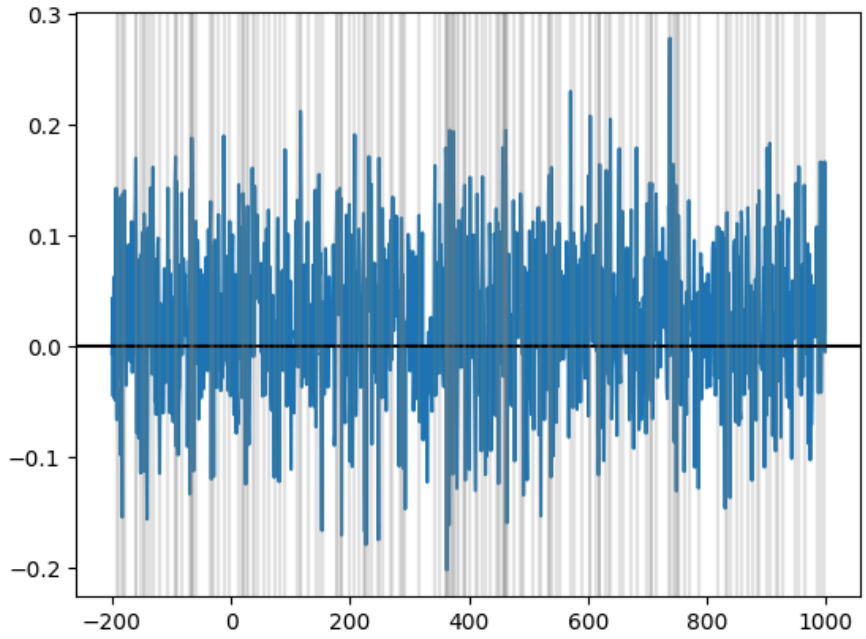
In [30]:
```
# 单样本t检验
t_vals, p_vals = ttest_1samp(data, 0, axis=0, alternative='greater')
# 打印出p_vals的shape：[1200]对应1200个时间点
print(np.shape(p_vals))
# 可视化统计检验后结果
# 阴影竖线代表显著的时间点
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, p_val in enumerate(p_vals):
    if p_val < 0.05:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```

```
(1200,)
```



非参数检验方法：置换检验 permutation test（未矫正）

In [31]:
```
# 使用NeuroRA的stuff模块下的permutation_test()函数进行置换检验
# 生成一个shape为[20]的全0向量
zeros = np.zeros([20])
# 初始化一个p_vals用于存储计算得到的p值
p_vals = np.zeros([1200])
# 逐时间点计算p值
for t in range(1200):
    p_vals[t] = permutation_test(data[:, t], zeros)
# 可视化统计检验后结果
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, p_val in enumerate(p_vals):
    if p_val < 0.05:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```

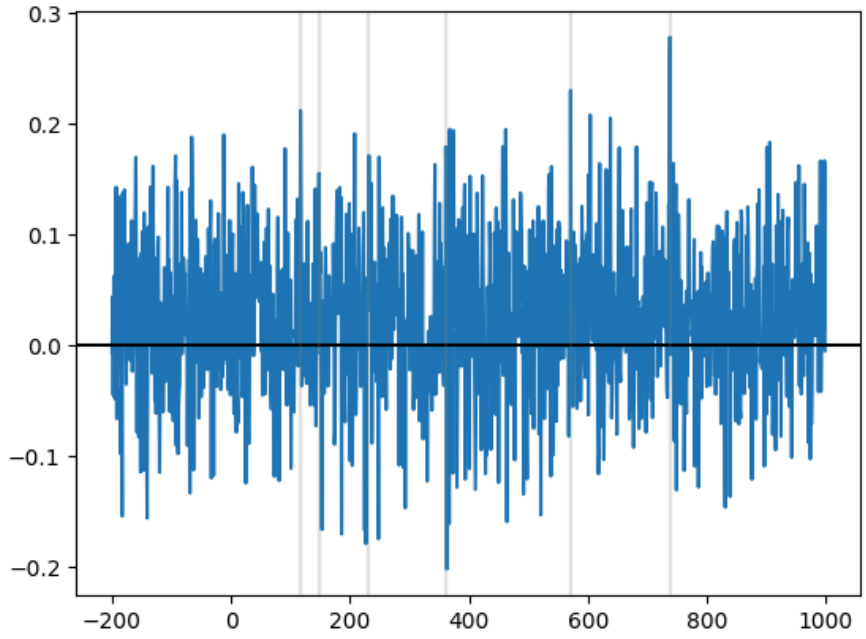


但若进行探索性分析时，希望找到有显著差异的时间点，这时需要对结果进行校正以避免多重比较带来的假阳性
常用的方法有以下三种：

- 控制族错误率（familywise error rate, FWER）的Bonferroni校正
- 控制错误发现率（False discovery rate, FDR）的FDR校正
- 基于簇的置换检验（cluster-based permutation test）

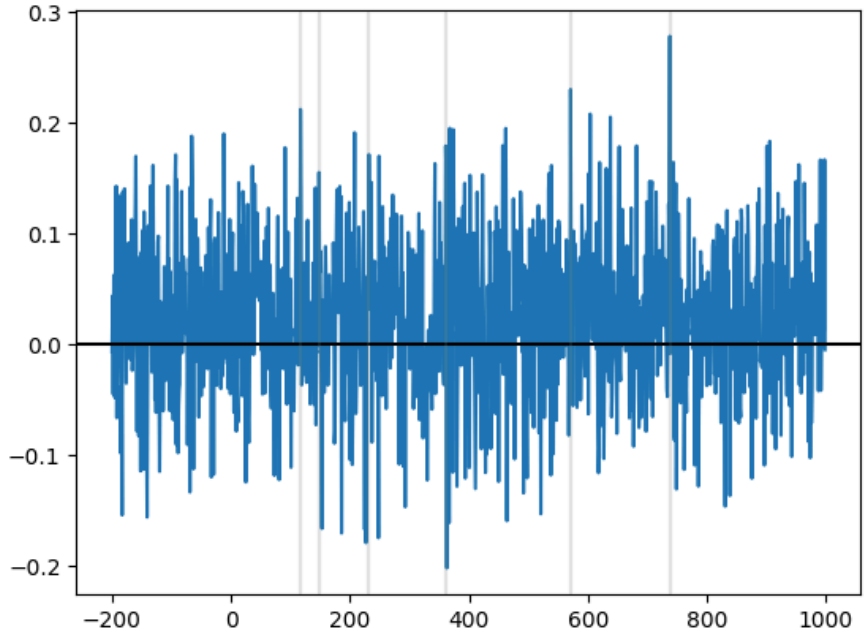
Bonferroni（FWER）矫正

In [32]:
```
# Bonferrorni校正只需要把p值乘以进行统计检验的次数
# 这里即乘以时间点数
p_bf_corrected_vals = p_vals*len(times)
# 可视化经过矫正的统计检验结果
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, p_val in enumerate(p_bf_corrected_vals):
    if p_val < 0.05:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```



FDR矫正

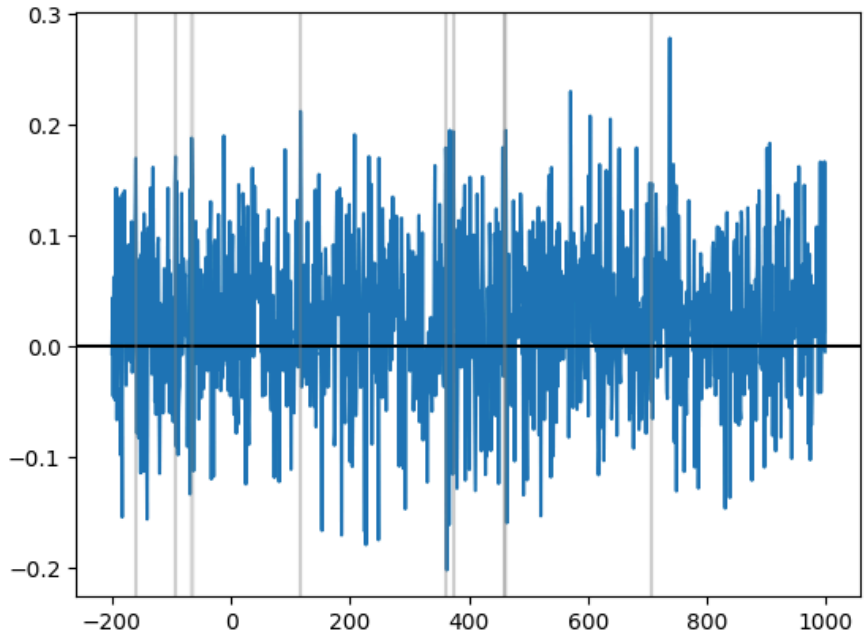
In [33]:
```
# FDR校正可以使用MNE中stats模块下的fdr_correlation()函数实现
# 其第一个返回值为是否通过矫正的布尔型数组（True为矫正后仍显著），第二个返回值为矫正后的p值
rejects, p_fdr_corrected_vals = fdr_correction(p_vals, alpha=0.05)
# 可视化经过矫正的统计检验结果
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, p_val in enumerate(p_fdr_corrected_vals):
    if p_val < 0.05:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```



基于簇的置换检验

In [34]:
```
# 这里时序（1维）数据的单样本单边Cluster-based permutation test
# 可以通过NeuroRA中stuff模块下clusterbased_permutation_1d_1samp_1sided()函数实现
# 其需要输入的是用来做统计的数据（即这里的data），输出是是矫正后是否显著的矩阵（1为显著的点）
# 这里先使用p<0.05的阈值来选取clusters，对应参数设置为p_threshold=0.05
# 再用p<0.05的阈值来进行基于cluster的矫正，对应参数为clusterp_threshold=0.05
rejects = clusterbased_permutation_1d_1samp_1sided(data, level=0,
                                                   p_threshold=0.05,
                                                   clusterp_threshold=0.05)
# 可视化经过矫正的统计检验结果
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, reject in enumerate(rejects):
    if reject == 1:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```

```
Permutation test
Calculating: [================================================================================================================] 100.00%
Cluster-based permutation test finished!
```



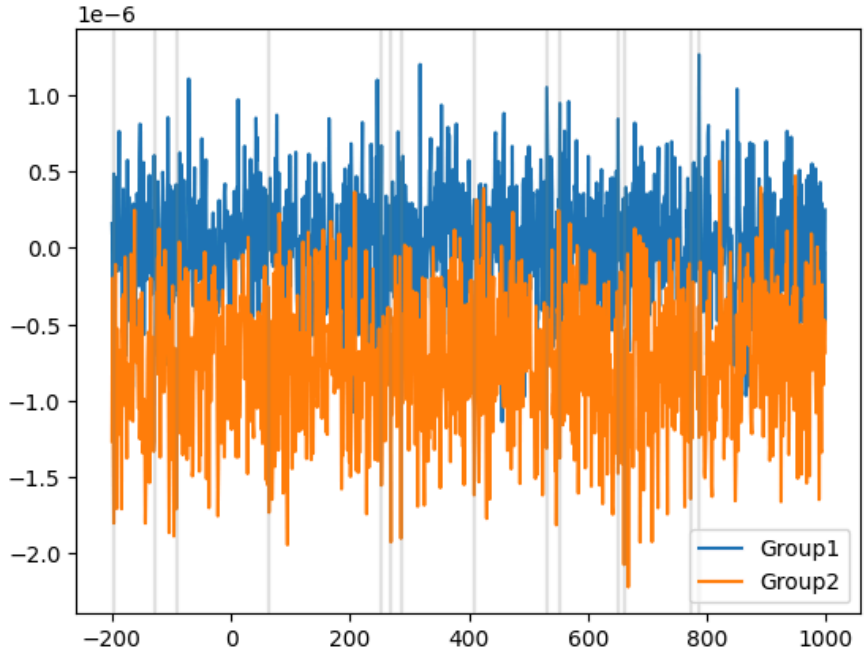
比较两条件/群组间是否有差异

当需要比较两个条件下的脑活动是否存在显著差异时
可以使用独立样本t检验(条件为被试间变量)、配对样本t检验（条件为被试内变量）、置换检验

参数检验方法：独立样本t检验（FDR 校正）

这里随机生成两组虚假脑电数据
假设它们分别为正常人组（对应data1）和病人组（对应data2）的数据
正常人组有20个被试，病人组有18个被试，epoch长度依然为1200ms（1200个时间点，从-200ms到1000ms）

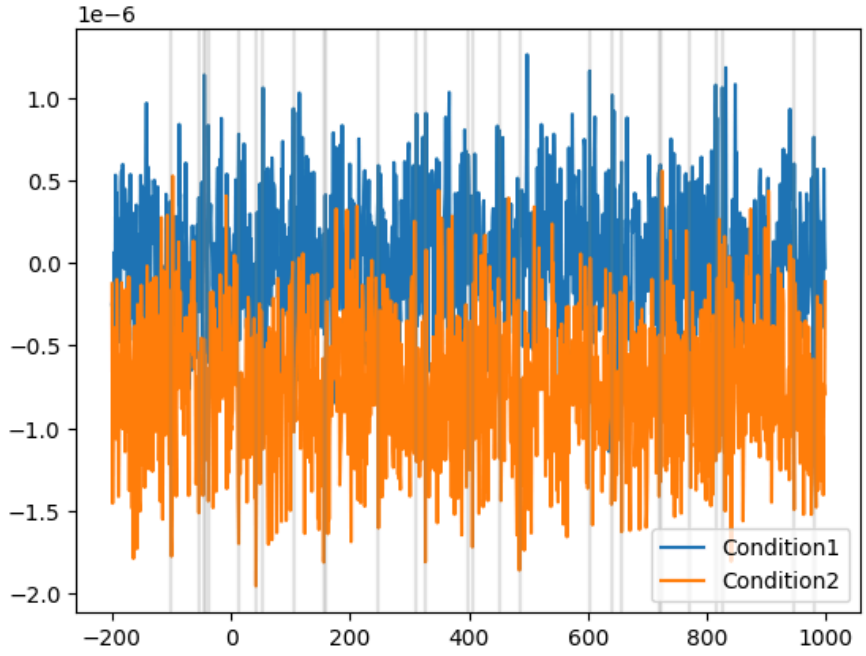
In [35]:
```
# 生成虚假（随机）数据
data1 = np.random.uniform(low=-3e-6, high=3e-6, size=(20, 1200))
data2 = np.random.uniform(low=-4e-6, high=2.5e-6, size=(18, 1200))
# 独立样本t检验
t_vals, p_vals = ttest_ind(data1, data2, axis=0)
# FDR矫正
rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=0.05)
# 可视化经过矫正的统计检验结果
plt.plot(times, np.average(data1, axis=0), label='Group1')
plt.plot(times, np.average(data2, axis=0), label='Group2')
for i, reject in enumerate(rejects):
    if reject == True:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.legend()
plt.show()
```



参数检验方法：配对样本t检验（FDR 校正）

这里随机生成两组虚假脑电数据
假设它们分别为同一组被试在不同条件下的数据（条件1对应data1，条件2对应data2）
被试数为20，epoch长度依然为1200ms（1200个时间点，从-200ms到1000ms）

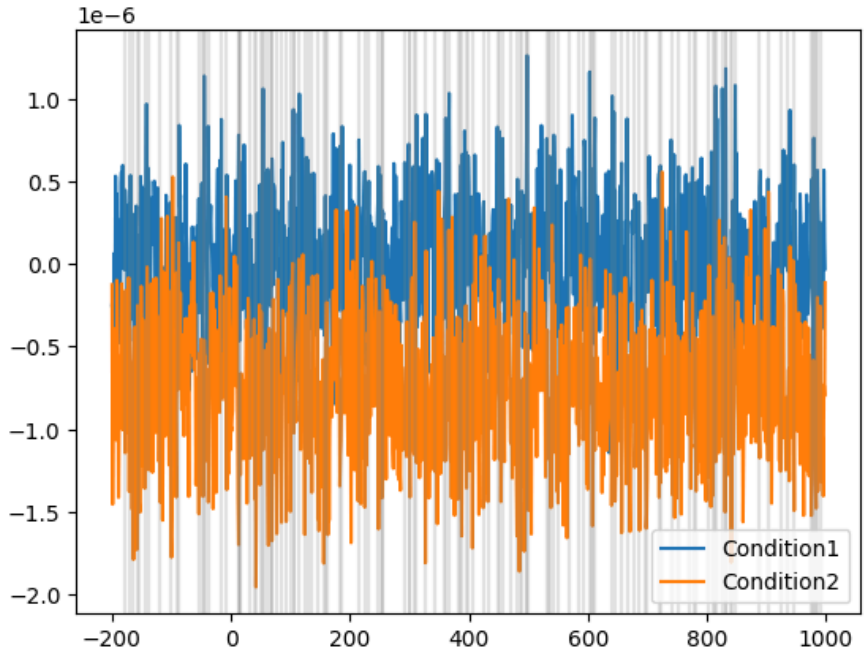
In [36]:
```
# 生成虚假（随机）数据
data1 = np.random.uniform(low=-3e-6, high=3e-6, size=(20, 1200))
data2 = np.random.uniform(low=-4e-6, high=2.5e-6, size=(20, 1200))
# 配对样本t检验
t_vals, p_vals = ttest_rel(data1, data2, axis=0)
# FDR矫正
rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=0.05)
# 可视化经过矫正后的统计检验结果
plt.plot(times, np.average(data1, axis=0), label='Condition1')
plt.plot(times, np.average(data2, axis=0), label='Condition2')
for i, reject in enumerate(rejects):
    if reject == True:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.legend()
plt.show()
```



非参数检验方法：置换检验（FDR矫正）

使用上例（配对样本t检验）的虚假数据示例

In [37]:
```
# 初始化一个p_vals用于存储计算得到的p值
p_vals = np.zeros([1200])
# 逐时间点计算p值
for t in range(1200):
    p_vals[t] = permutation_test(data1[:, t], data2[:, t])
# FDR矫正
rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=0.05)
# 可视化经过矫正的统计检验结果
plt.plot(times, np.average(data1, axis=0), label='Condition1')
plt.plot(times, np.average(data2, axis=0), label='Condition2')
for i, reject in enumerate(rejects):
    if reject == 1:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.legend()
plt.show()
```

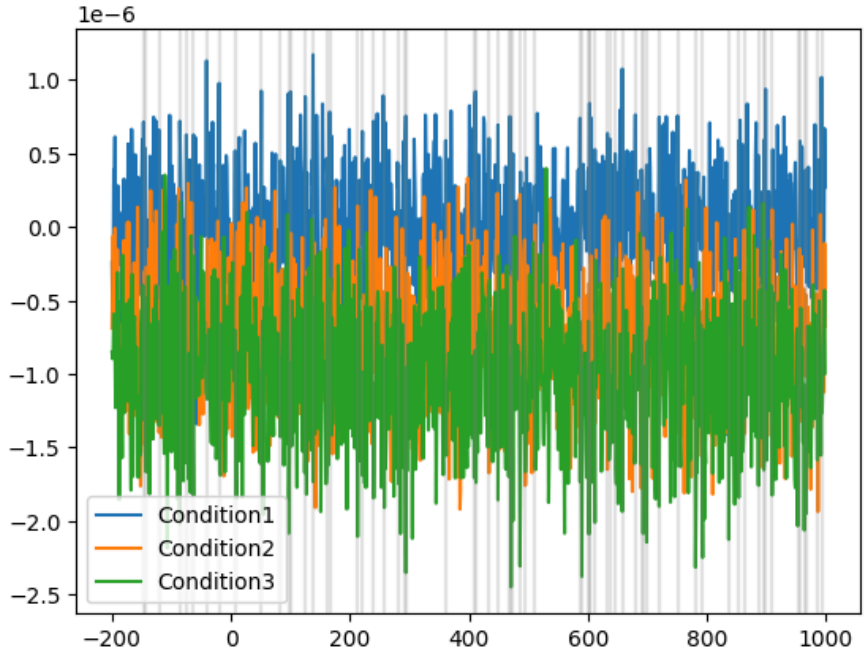


对比多个条件/群组间有无显著差异

参数检验方法：单因素方差分析（FDR矫正）

这里，类似前例，生成同一组被试在三种不同实验条件下的虚假脑电数据（分别对应data1、data2和data3）
被试量为20，epoch长度为1200ms（1200个时间点，从-200ms到1000ms）

In [38]:
```
# 生成虚假（随机）数据
data1 = np.random.uniform(low=-3e-6, high=3e-6, size=(20, 1200))
data2 = np.random.uniform(low=-4e-6, high=2.5e-6, size=(20, 1200))
data3 = np.random.uniform(low=-4.5e-6, high=2.5e-6, size=(20, 1200))
# 单因素F检验
f_vals, p_vals = f_oneway(data1, data2, data3, axis=0)
# FDR矫正
rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=0.05)
# 可视化经过矫正的统计检验结果
plt.plot(times, np.average(data1, axis=0), label='Condition1')
plt.plot(times, np.average(data2, axis=0), label='Condition2')
plt.plot(times, np.average(data3, axis=0), label='Condition3')
for i, reject in enumerate(rejects):
    if reject == True:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.legend()
plt.show()
```

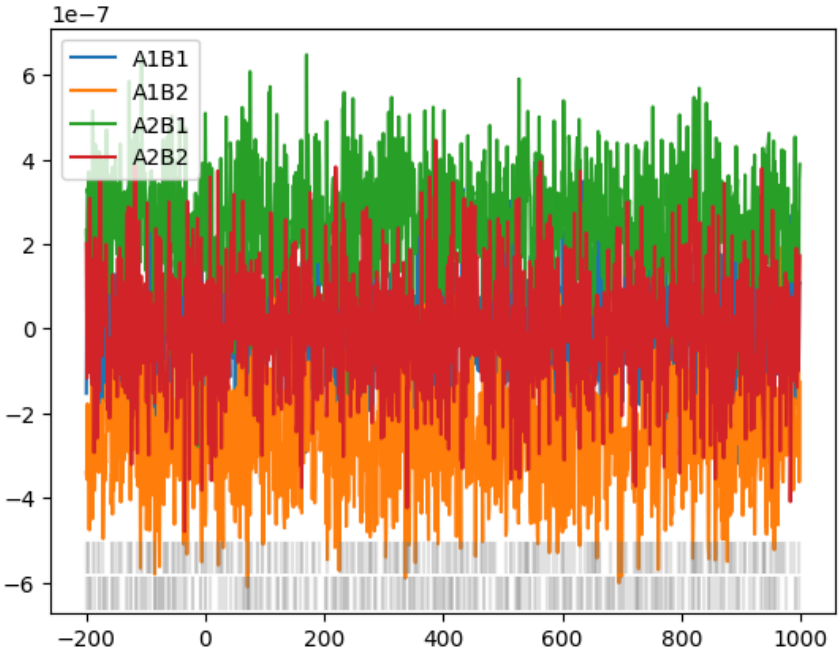


多因素设计：两因素的主效应与交互效应

参数检验方法：2×2重复测量方差分析（FDR矫正）

假设有两个被试内变量：A和B
A有两个水平：A1, A2; B有两个水平：B1, B2
这里类似上面，随机生一组被试4种条件下（A1B1、A1B2、A2B1、A2B2）的虚假脑电数据
被试数为200，epoch长度为1200ms（1200个时间点，从-200ms到1000ms）

In [39]:
```
# 生成虚假（随机）数据
data_A1B1 = np.random.uniform(low=-3e-6, high=3e-6, size=(200, 1200))
data_A1B2 = np.random.uniform(low=-3.5e-6, high=3e-6, size=(200, 1200))
data_A2B1 = np.random.uniform(low=-3e-6, high=3.5e-6, size=(200, 1200))
data_A2B2 = np.random.uniform(low=-3.5e-6, high=3.5e-6, size=(200, 1200))
# 首先对数据进行reshape方便后续合并
reshaped_A1B1 = data_A1B1.reshape(200, 1, 1200)
reshaped_A1B2 = data_A1B2.reshape(200, 1, 1200)
reshaped_A2B1 = data_A2B1.reshape(200, 1, 1200)
reshaped_A2B2 = data_A2B2.reshape(200, 1, 1200)
#把数据按照两个因素的顺序（A1B1、A1B2、A2B1、A2B2）合并
data_combine = np.concatenate((reshaped_A1B1, reshaped_A1B2,
                               reshaped_A2B1, reshaped_A2B2), axis=1)
# 设置变量水平
factor_levels = [2, 2]
# 使用MNE的f_mway_rm函数进行2×2方差分析
# 变量A的主效应
f_main_A, p_main_A = f_mway_rm(data_combine, factor_levels, effects='A')
# 变量B的主效应
f_main_B, p_main_B = f_mway_rm(data_combine, factor_levels, effects='B')
# 交互效应
f_inter, p_interaction = f_mway_rm(data_combine, factor_levels, effects='A:B')
# FDR矫正
rejects_A, p_main_A = fdr_correction(p_main_A, alpha=0.05)
rejects_B, p_main_B = fdr_correction(p_main_B, alpha=0.05)
rejects_inter, p_interaction = fdr_correction(p_interaction, alpha=0.05)
# 可视化经过矫正的统计检验结果
# 图片下方三行灰色竖线，有下至上分别代表A主效应、B主效应和交互效应显著的时间点
plt.plot(times, np.average(data_A1B1, axis=0), label='A1B1')
plt.plot(times, np.average(data_A1B2, axis=0), label='A1B2')
plt.plot(times, np.average(data_A2B1, axis=0), label='A2B1')
plt.plot(times, np.average(data_A2B2, axis=0), label='A2B2')
for i in range(1200):
    if p_main_A[i] < 0.05:
        plt.axvline(x=times[i], ymin=0.01, ymax=0.06, color='grey', alpha=0.2)
    if p_main_B[i] < 0.05:
        plt.axvline(x=times[i], ymin=0.07, ymax=0.12, color='grey', alpha=0.2)
    if p_interaction[i] < 0.05:
        plt.axvline(x=times[i], ymin=0.13, ymax=0.18, color='grey', alpha=0.2)
plt.legend()
plt.show()
```


1e−7
A1B1
A1B2
A2B1
A2B2
6
4
2
0
−2
−4
−6
−200
0
200
400
600
800
1000

第三章: 多被试分析

多被试分析的这一章分成了以下三个部分：

- **第一节: 批处理读取与存储数据**
- **第二节: 事件相关电位分析**
- **第三节: 时频分析**

下载并导入需要用到的Python包

In [1]:
```
! pip install gdown

import numpy as np
import os
import gdown
import zipfile
import h5py
import scipy.io as sio
import matplotlib.pyplot as plt
from scipy.stats import ttest_1samp, ttest_rel
from mne.stats import fdr_correction, f_mway_rm
from neurora.stuff import clusterbased_permutation_1d_1samp_1sided, \
                          permutation_test, \
                          clusterbased_permutation_2d_1samp_2sided, \
                          clusterbased_permutation_2d_2sided
from mne.time_frequency import tfr_array_morlet
```

```
Requirement already satisfied: gdown in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (5.1.
0)
Requirement already satisfied: beautifulsoup4 in /Users/zitonglu/anaconda3/lib/python3.11/site-packa
ges (from gdown) (4.12.2)
Requirement already satisfied: filelock in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (f
rom gdown) (3.9.0)
Requirement already satisfied: requests[socks] in /Users/zitonglu/anaconda3/lib/python3.11/site-pack
ages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
gdown) (4.65.0)
Requirement already satisfied: soupsieve>1.2 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from beautifulsoup4->gdown) (2.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/zitonglu/anaconda3/lib/python3.11/
site-packages (from requests[socks]->gdown) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from requests[socks]->gdown) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests[socks]->gdown) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests[socks]->gdown) (2023.7.22)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /Users/zitonglu/anaconda3/lib/python3.11/si
te-packages (from requests[socks]->gdown) (1.7.1)
```

第一节 - 批处理读取与存储数据

预处理过的Demo数据1：

（你将会在第二节和第三节中使用）
原数据集基于Wakeman&Henson于2015年发表在Scientific Data上的文章《A multi-subject, multi-modal human neuroimaging dataset》
在这个实验中，共有三种类型的面孔，分别为熟悉面孔、不熟悉面孔和乱相面孔，各150张图像，共450张刺激图像
被试佩戴脑电进行简单的知觉任务，800-1000ms不定的刺激阶段+1700ms的延迟+400-600ms不定的ITI
每张图片会被看到两次，第二次50%的刺激图像可能是紧接着第一次的下一个试次马上又重复观看同一图像，50%间隔较多试次后再呈现
这里仅提取前8个被试对多张熟悉面孔图像第一次观看和对这其中一部分图像紧接着下一个试次又重复观看的试次的脑电数据

In [2]:
```
# 下载Demo数据1
```

```
data_dir = "data/"
if not os.path.exists(data_dir):
  os.makedirs(data_dir)

# 从Google Drive下载
url = "https://drive.google.com/file/d/1hsPmIFod3c7ZR0YdyO8woqUfqO_3pZtR/view?usp=sharing"
filename = "demo_data1"
filepath = data_dir + filename + ".zip"

# 下载数据
gdown.download(url=url, output=filepath, quiet=False, fuzzy=True)
print("Download completes!")
# 解压数据
with zipfile.ZipFile(filepath, 'r') as zip:
    zip.extractall(data_dir)
print("Unzip completes!")

# 也可以通过百度网盘下载
# 链接:https://pan.baidu.com/s/173Hjnt-JOw0BpIm5OEx9kg  密码:2u7y
# 下载后解压，并移动到data文件夹下
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1hsPmIFod3c7ZR0YdyO8woqUfqO_3pZtR
From (redirected): https://drive.google.com/uc?id=1hsPmIFod3c7ZR0YdyO8woqUfqO_3pZtR&confirm=t&uuid=64c004fa-0289-4fe3-9cc3-c66c63609f2e
To: /Users/zitonglu/Downloads/Python-EEG-Handbook-master/data/demo_data1.zip
100%|████████████████████████████████████████| 242M/242M [00:28<00:00, 8.43MB/s]
Download completes!
Unzip completes!
```

以sub1为例，'sub1.mat'为对熟悉面孔图像第一次观看的脑电数据，'sub1_rep.mat'为对熟悉面孔马上重复观看第二次的脑电数据

前一情况试次数为后一情况试次数一倍，数据为进行过预处理（0.1-30Hz滤波）并分好段之后的数据

数据中，导联数为74（其中脑电导联为70个，第61， 62， 63， 64个导联为眼动导联），采样率为250Hz

从刺激呈现的前0.5s到刺激呈现后的1.5s，每个试次包含500个时间点

批处理读取Demo数据1并存成.h5文件

In [3]:
```
# 对8个被试的数据进行遍历
for sub in range(8):

    # 获取每个被试的两个条件下的.mat文件地址
    subdata_first_path = 'data/demo_data1/sub' + str(sub + 1) + '_first.mat'
    subdata_rep_path = 'data/demo_data1/sub' + str(sub + 1) + '_rep.mat'

    # 通过.mat文件提取数据
    # 数据shape为[n_channels, n_times, n_trials]
    subdata_first = sio.loadmat(subdata_first_path)['data']
    subdata_rep = sio.loadmat(subdata_rep_path)['data']

    # 由于导联中有四个导联是眼电导联，需要进行删去
    subdata_first = np.delete(np.array(subdata_first), [60, 61, 62, 63], axis=0)
    subdata_rep = np.delete(np.array(subdata_rep), [60, 61, 62, 63], axis=0)

    # 打印被试编号及熟悉面孔第一看到和紧接着重复看到两条件下的脑电矩阵shape
    print('sub' + str(sub + 1))
    print(subdata_first.shape)
    print(subdata_rep.shape)

    # 将每个被试两刺激条件下的脑电数据（矩阵形式）以‘data’和'data_rep'为Keys存在一个.h5文件里
    f = h5py.File('data/demo_data1/sub' + str(sub + 1) + '.h5', 'w')
    f.create_dataset('data_first', data=subdata_first)
    f.create_dataset('data_rep', data=subdata_rep)
    f.close()
```

```
sub1
(70, 500, 150)
(70, 500, 78)
sub2
(70, 500, 150)
(70, 500, 79)
sub3
(70, 500, 150)
(70, 500, 74)
sub4
(70, 500, 150)
(70, 500, 68)
sub5
(70, 500, 150)
(70, 500, 67)
sub6
(70, 500, 150)
(70, 500, 70)
sub7
(70, 500, 150)
(70, 500, 77)
sub8
(70, 500, 150)
(70, 500, 70)
```

第二节 - 事件相关电位分析

这里，我们将使用Demo数据1为例

我们将分别对实验中熟悉面孔首次观看与熟悉面孔立即重复观看两条件下的数据进行ERP的统计分析与可视化

读取数据与叠加平均

此例里以第50个导联的ERP示例，选取-200ms到1000ms部分

In [4]:
```
# 初始化2个变量分别存储每个被试两条件下的ERP
# 8 - n_subjects, 300 - n_times  (-200ms到1000ms)
erp_first = np.zeros([8, 300])
erp_rep = np.zeros([8, 300])

# 逐被试迭代
for sub in range(8):

    # 读取该被试的数据
    with h5py.File('data/demo_data1/sub' + str(sub + 1) + '.h5', 'r') as f:
        subdata_first = np.array(f['data_first'])
        subdata_rep = np.array(f['data_rep'])
        f.close()
    # subdata和subdata_rep的shape均为[n_channels, n_times, n_trials]

    # 选取第50个导联的数据，平均试次，并取-200ms到1000ms
    erp_first[sub] = np.average(subdata_first[49], axis=1)[75:375]
    erp_rep[sub] = np.average(subdata_rep[49], axis=1)[75:375]

    # 基线校正，取基线为-100到0ms
    erp_first[sub] = erp_first[sub] - np.average(erp_first[sub, 25:50])
    erp_rep[sub] = erp_rep[sub] - np.average(erp_rep[sub, 25:50])
```

统计分析与结果可视化

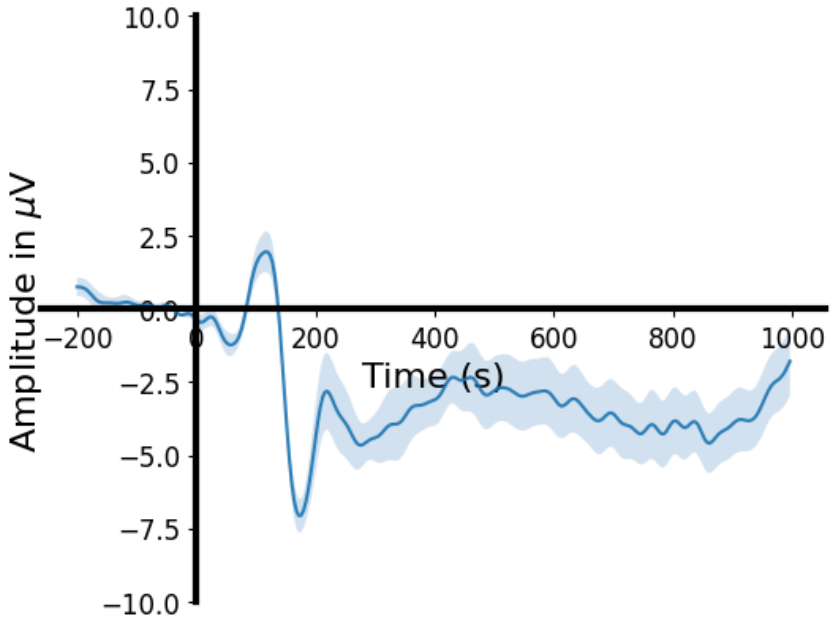
定义一个可视化ERP结果的函数 - plot_erp_results()

In [5]:
```
def plot_erp_results(erp, times, ylim=[-10, 10], labelpad=0):

    """
    参数:
        erp: shape为[n_subs, n_times]的矩阵，对应每个被试的ERP
        times: shape为[n_times]的array，代表时间点（对应x轴的时间范围及时间点）
        ylim: Y轴范围，默认[-10, 10]
        labelpad: Y轴标签离坐标轴的距离，默认0
    """

    n_subjects = np.shape(erp)[0]
```

```
    # 平均ERPs
    avg = np.average(erp, axis=0)
    # 计算逐时间点的SEM
    err = np.std(erp, axis=0, ddof=0)/np.sqrt(n_subjects)

    ax = plt.gca()
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_linewidth(3)
    ax.spines["left"].set_position(("data", 0))
    ax.spines["bottom"].set_linewidth(3)
    ax.spines['bottom'].set_position(('data', 0))

    # 绘制ERP
    plt.fill_between(times, avg+err, avg-err, alpha=0.2)
    plt.plot(times, avg, alpha=0.9)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.ylabel(r'Amplitude in $\mu$V', fontsize=16, labelpad=labelpad)
    plt.xlabel('Time (s)', fontsize=16)
    plt.ylim(ylim[0], ylim[1])
    plt.show()
```

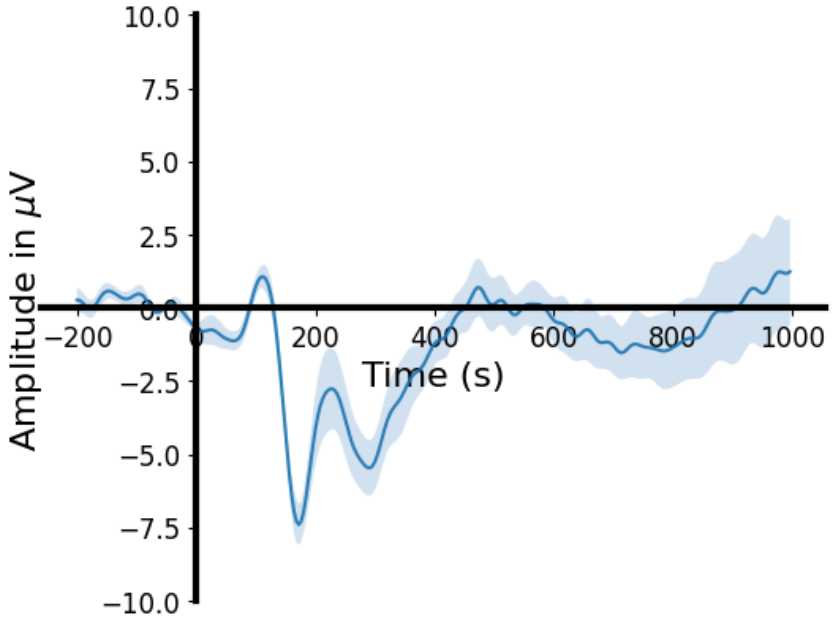
绘制首次观看熟悉面孔条件下的ERP

In [6]:
```
times = np.arange(-200, 1000, 4)
plot_erp_results(erp_first, times, labelpad=25)
```



绘制立即重复观看熟悉面孔条件下的ERP

In [7]:
```
plot_erp_results(erp_rep, times, labelpad=25)
```



定义一个可视化两条件下ERP结果的函数 - plot_erp_2cons_results()

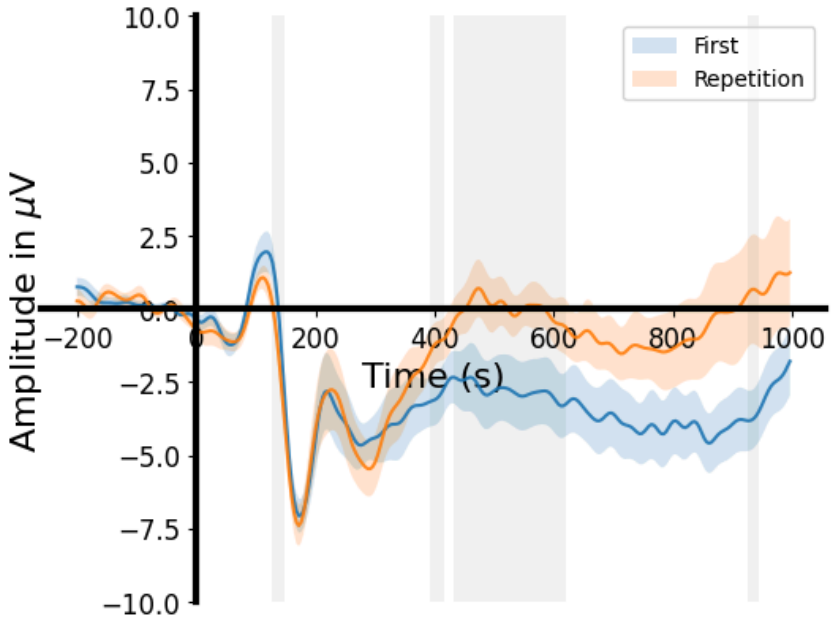
In [8]:

```
def plot_erp_2cons_results(erp1, erp2, times, con_labels=['Condition1', 'Condition2'], ylim=[-10, 10
                      p_threshold=0.05, labelpad=0):

    """
    参数:
        erp1: shape为[n_subs, n_times]的矩阵，对应每个被试条件1下的ERP
        erp2: shape为[n_subs, n_times]的矩阵，对应每个被试条件2下的ERP
        con_labels: 一个List或array，[条件1名称，条件2名称]，
                    默认为['Condition1', 'Condition2']
        times: shape为[n_times]的array，代表时间点（对应x轴的时间范围及时间点）
        con_labels: 条件标签，默认['Condition 1', 'Condition 2']
        ylim: Y轴范围，默认[-10, 10]
        p_threshold : 一个浮点型数字，默认为0.05，代表p值的阈值
        labelpad: Y轴标签离坐标轴的距离，默认0
    """

    n_subjects = np.shape(erp1)[0]

    # 平均ERPs
    avg1 = np.average(erp1, axis=0)
    avg2 = np.average(erp2, axis=0)
    # calcualte the SEM for each time-point
    err1 = np.std(erp1, axis=0, ddof=0)/np.sqrt(n_subjects)
    err2 = np.std(erp2, axis=0, ddof=0)/np.sqrt(n_subjects)

    # 统计分析
    t_vals, p_vals = ttest_rel(erp1, erp2, axis=0)
    # FDR矫正
    # rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=p_threshold)

    ax = plt.gca()
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_linewidth(3)
    ax.spines["left"].set_position(("data", 0))
    ax.spines["bottom"].set_linewidth(3)
    ax.spines['bottom'].set_position(('data', 0))

    # 标记显著时间窗
    tstep = (times[-1]-times[0])/len(times)
    for i, p_val in enumerate(p_vals):
        if p_val < 0.05:
            plt.fill_between([times[i], times[i]+tstep], [ylim[1]], [ylim[0]],
                             facecolor='gray', alpha=0.1)

    # 绘制带统计结果的ERP
    plt.fill_between(times, avg1+err1, avg1-err1, alpha=0.2, label=con_labels[0])
    plt.fill_between(times, avg2+err2, avg2-err2, alpha=0.2, label=con_labels[1])
    plt.plot(times, avg1, alpha=0.9)
    plt.plot(times, avg2, alpha=0.9)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
```

```
    plt.ylim(ylim[0], ylim[1])
    plt.ylabel(r'Amplitude in $\mu$V', fontsize=16, labelpad=labelpad)
    plt.xlabel('Time (s)', fontsize=16)

    plt.legend()
    plt.show()
```

统计并绘制首次观看与立即重复观看熟悉面孔两条件下的ERP

In [9]:
```
plot_erp_2cons_results(erp_first, erp_rep, times, con_labels=['First', 'Repetition'],
                       p_threshold=0.05, labelpad=25)
```



第三节 - 时频分析

我们分别对实验中首次观看熟悉面孔与立即重复观看熟悉面孔两条件下的数据进行时频分析

读取数据并进行时频分析

In [10]:
```
# 初始化2个变量分别存储每个被试两条件下的时频分析结果
# 8 - n_subjects, 70 - n_channels, 14 - n_freqs,  500 - n_times
tfr_first = np.zeros([8, 70, 14, 500])
tfr_rep = np.zeros([8, 70, 14, 500])

# 逐被试迭代
for sub in range(8):

    # 读取该被试的数据
    with h5py.File('data/demo_data1/sub' + str(sub + 1) + '.h5', 'r') as f:
        subdata_first = np.array(f['data_first'])
        subdata_rep = np.array(f['data_rep'])
        f.close()
    # 数据的shape从[n_channels, n_times, n_trials]转换成[n_trials, n_channels, n_times]
    subdata_first = np.transpose(subdata_first, (2, 0, 1))
    subdata_rep = np.transpose(subdata_rep, (2, 0, 1))

    # 设定一些时频分析的参数
    # 频段选取4-30Hz
    freqs = np.arange(4, 32, 2)
    n_cycles = freqs / 2.
    # 时频分析
    # 使用MNE的time_frequency模块下的tfr_arrayy_morlet()函数
    # 其输入为[n_epochs, n_channels, n_times]的array
    # 同时接着依次传入数据采样率、计算频率、周期数和输出数据类型
    subtfr_first = tfr_array_morlet(subdata_first, 250, freqs, n_cycles, output='power')
    subtfr_rep = tfr_array_morlet(subdata_rep, 250, freqs, n_cycles, output='power')
    # 此时返回的tfr的shape为[n_trials, n_channels, n_freqs, n_times]
    # 这里，对试次与导联维度平均传入tfr或tfr_rep变量中
    tfr_first[sub] = np.average(subtfr_first, axis=0)
    tfr_rep[sub] = np.average(subtfr_rep, axis=0)
    # 基线校正，这里使用'logratio'方法，即除以基线均值并取log
```

```
    # 取基线为-100到0ms
    for chl in range(70):
        for freq in range(len(freqs)):
            tfr_first[sub, chl, freq] = 10 * np.log10(tfr_first[sub, chl, freq] /
                                    np.average(tfr_first[sub, chl, freq, 100:125]))
            tfr_rep[sub, chl, freq] = 10 * np.log10(tfr_rep[sub, chl, freq] /
                                        np.average(tfr_rep[sub, chl, freq, 100:125]))
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    3.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    3.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    3.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    2.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    2.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    2.9s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    3.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    2.9s finished
```

这里以第50个导联的两条件下-200ms到1000ms的时频分析结果作为后面统计分析与可视化的示例

In [11]:
```
tfr_first_No50 = tfr_first[:, 49, :, 75:375]
tfr_rep_No50 = tfr_rep[:, 49, :, 75:375]
```

统计分析与结果可视化

定义一个绘制单条件时频分析结果的函数 - plot_tfr_results().

这一部分包含统计分析与可视化的功能，并且对结果进行了cluster-based permutation test

In [12]:
```
def plot_tfr_results(tfr, freqs, times, p=0.01, clusterp=0.05, clim=[-4, 4]):

    """
    参数:
        tfr : shape为[n_subs, n_freqs, n_times]的矩阵，代表时频分析的结果
        freqs : shape为[n_freqs]的array，代表时频分析的频率（对应y轴的频率范围及频率点）
        times : shape为[n_times]的array，代表时频分析的时间点（对应x轴的时间范围及时间点）
        p : 一个浮点型数字，默认为0.01，代表p值的阈值
        clusterp : 一个浮点型数字，默认为0.05，代表cluster层面p值的阈值
        clim : 一个List或array，[最小值，最大值]，默认为[-4, 4]，代表颜色条的上下边界
    """

    n_freqs = len(freqs)
    n_times = len(times)

    # 统计分析
    # 注意：由于进行了cluster-based permutation test，需要运行较长时间
    # 这里使用NeuroRA的stuff模块下的clusterbased_permutation_2d_1samp_2sided()函数
    # 其返回的stats_results为一个shape为[n_freqs, n_times]的矩阵
    # 该矩阵中不显著的点的值为0，显著大于0的点的值为1，显著小于0的点的值为-1
    # 这里iter设成100是为了示例运行起来快一些，建议1000
    stats_results = clusterbased_permutation_2d_1samp_2sided(tfr, 0,
                                                             p_threshold=p,
                                                             clusterp_threshold=clusterp,
                                                             iter=100)

    # 时频分析结果可视化
    fig, ax = plt.subplots(1, 1)
    # 勾勒显著性区域
    padsats_results = np.zeros([n_freqs + 2, n_times + 2])
    padsats_results[1:n_freqs+1, 1:n_times + 1] = stats_results
    x = np.concatenate(([times[0]-1], times, [times[-1]+1]))
    y = np.concatenate(([freqs[0]-1], freqs, [freqs[-1]+1]))
    X, Y = np.meshgrid(x, y)
    ax.contour(X, Y, padsats_results, [0.5], colors="red", alpha=0.9,
               linewidths=2, linestyles="dashed")
    ax.contour(X, Y, padsats_results, [-0.5], colors="blue", alpha=0.9,
               linewidths=2, linestyles="dashed")
    # 绘制时频结果热力图
    im = ax.imshow(np.average(tfr, axis=0), cmap='RdBu_r', origin='lower',
                   extent=[times[0], times[-1], freqs[0], freqs[-1]], clim=clim)
    ax.set_aspect('auto')
    cbar = fig.colorbar(im)
    cbar.set_label('dB', fontsize=12)
    ax.set_xlabel('Time (ms)', fontsize=16)
    ax.set_ylabel('Frequency (Hz)', fontsize=16)
    plt.show()
```

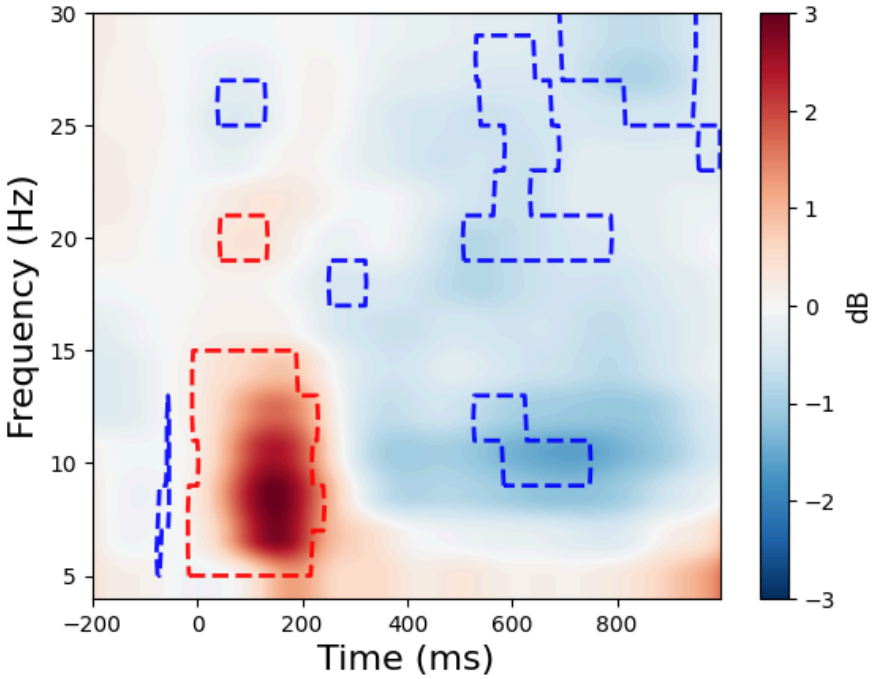
In [13]:
```
print(tfr_first.shape)
```

```
(8, 70, 14, 500)
```

统计并绘制首次观看熟悉面孔条件下的时频分析结果

In [14]:
```
freqs = np.arange(4, 32, 2)
times = np.arange(-200, 1000, 4)
plot_tfr_results(tfr_first_No50, freqs, times, p=0.05, clusterp=0.05, clim=[-3, 3])
```

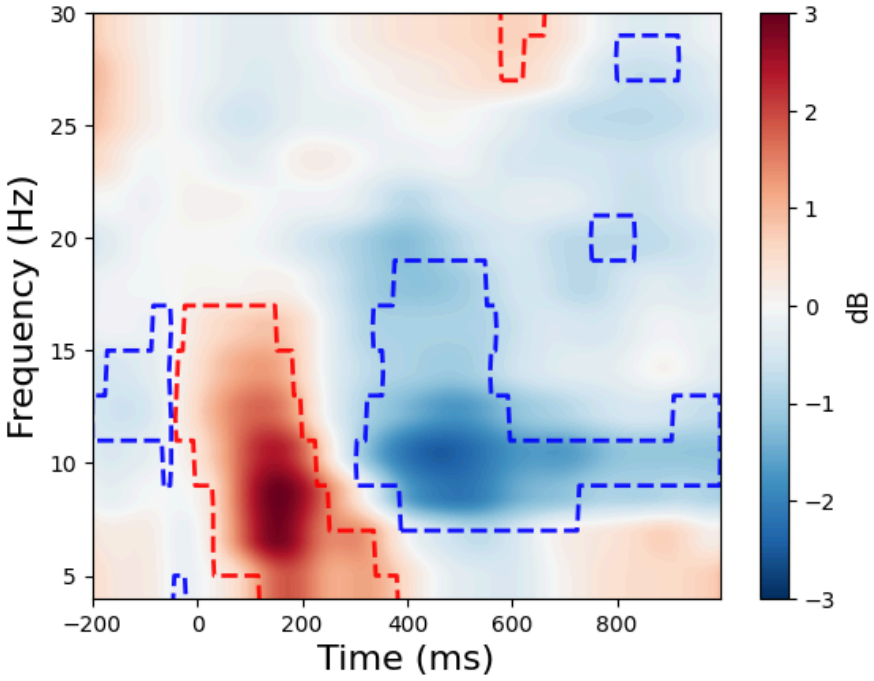
```
Permutation test

Side 1 begin:
Calculating: [============================================================================================================] 100.00%
Side 1 finished!

Side 2 begin:

Calculating: [============================================================================================================] 100.00%
Side 2 finished!

Cluster-based permutation test finished!
```



统计并绘制立即重复观看熟悉面孔条件下的时频分析结果

In [15]:
```
plot_tfr_results(tfr_rep_No50, freqs, times, p=0.05, clusterp=0.05, clim=[-3, 3])
```

```
Permutation test

Side 1 begin:
Calculating: [============================================================================================================] 100.00%
Side 1 finished!

Side 2 begin:

Calculating: [============================================================================================================] 100.00%
Side 2 finished!

Cluster-based permutation test finished!
```



定义一个绘制两条件时频分析结果差异结果的函数 - plot_tfr_diff_results()

这里的差异用tfr1 - tfr2计算得到

该函数包含统计分析与可视化的功能，并且结果进行了cluster-based permutation test

In [16]:

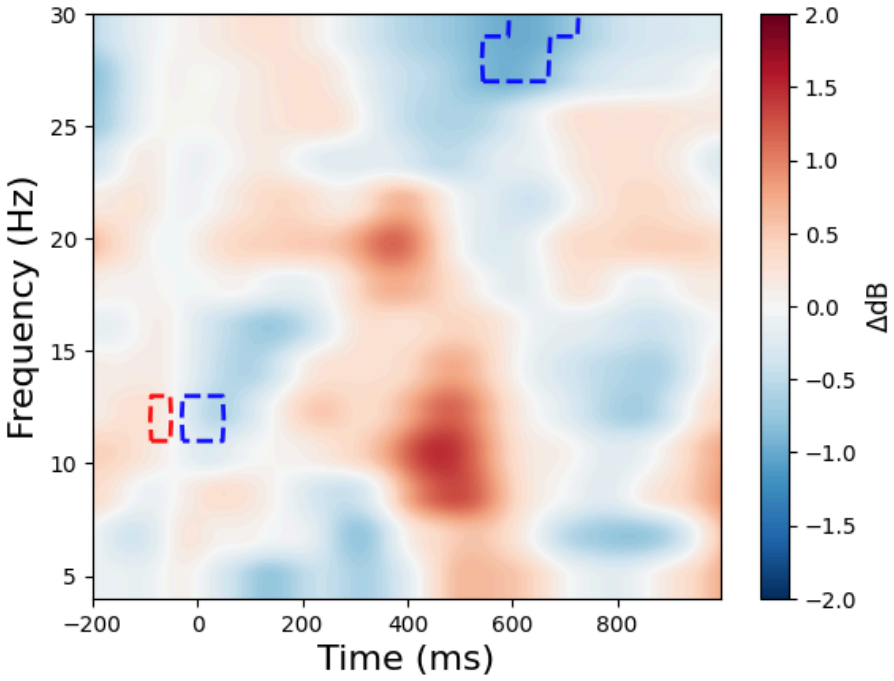
```
def plot_tfr_diff_results(tfr1, tfr2, freqs, times, p=0.01, clusterp=0.05,
                          clim=[-2, 2]):

    """
    参数:
        tfr1 : shape为[n_subs, n_freqs, n_times]的矩阵，代表条件1下时频分析的结果
        tfr2 : shape为[n_subs, n_freqs, n_times]的矩阵，代表条件2下时频分析的结果
        freqs : shape为[n_freqs]的array，代表时频分析的频率（对应y轴的频率范围及频率点）
        times : shape为[n_times]的array，代表时频分析的时间点（对应x轴的时间范围及时间点）
        p : 一个浮点型数字，默认为0.01，代表p值的阈值
        clusterp : 一个浮点型数字，默认为0.05，代表cluster层面p值的阈值
        clim : 一个List或array，[最小值，最大值]，默认为[-2, 2]，代表颜色条的上下边界
    """

    n_freqs = len(freqs)
    n_times = len(times)

    # 统计分析
    # 注意：由于进行了cluster-based permutation test，需要运行较长时间
    # 这里使用NeuroRA的stuff模块下的clusterbased_permutation_2d_2sided()函数
    # 其返回的stats_results为一个shape为[n_freqs, n_times]的矩阵
    # 该矩阵中不显著的点的值为0，条件1显著大于条件2的点的值为1，条件1显著小于条件2的点的值为-1
    # 这里iter设成100是为了示例运行起来快一些，建议1000
    stats_results = clusterbased_permutation_2d_2sided(tfr1, tfr2,
                                                       p_threshold=p,
                                                       clusterp_threshold=clusterp,
                                                       iter=100)

    # 计算tfr差异
    tfr_diff = tfr1 - tfr2

    # 时频分析结果可视化
    fig, ax = plt.subplots(1, 1)
    # 勾勒显著性区域
    padsats_results = np.zeros([n_freqs + 2, n_times + 2])
    padsats_results[1:n_freqs + 1, 1:n_times + 1] = stats_results
    x = np.concatenate(([times[0]-1], times, [times[-1]+1]))
    y = np.concatenate(([freqs[0]-1], freqs, [freqs[-1]+1]))
    X, Y = np.meshgrid(x, y)
    ax.contour(X, Y, padsats_results, [0.5], colors="red", alpha=0.9,
               linewidths=2, linestyles="dashed")
    ax.contour(X, Y, padsats_results, [-0.5], colors="blue", alpha=0.9,
               linewidths=2, linestyles="dashed")
    # 绘制时频结果热力图
    im = ax.imshow(np.average(tfr_diff, axis=0), cmap='RdBu_r', origin='lower',
                   extent=[times[0], times[-1], freqs[0], freqs[-1]], clim=clim)
```

```
    ax.set_aspect('auto')
    cbar = fig.colorbar(im)
    cbar.set_label('$\Delta$dB', fontsize=12)
    ax.set_xlabel('Time (ms)', fontsize=16)
    ax.set_ylabel('Frequency (Hz)', fontsize=16)
    plt.show()
```

统计并绘制首次观看与立即重复观看熟悉面孔两条件下的时频分析结果差异

In [17]:
```
freqs = np.arange(4, 32, 2)
times = np.arange(-200, 1000, 4)
plot_tfr_diff_results(tfr_first_No50, tfr_rep_No50, freqs, times,
                      p=0.05, clusterp=0.05, clim=[-2, 2])
```

```
Permutation test

Side 1 begin:
Calculating: [============================================================================================================] 100.00%
Side 1 finished!

Side 2 begin:

Calculating: [============================================================================================================] 100.00%
Side 2 finished!

Cluster-based permutation test finished!
```



第四章: 高级脑电数据分析

高级脑电数据分析的这一章分成了以下四个部分：

- **第一节: 批处理读取与存储数据**
- **第二节: 基于分类的脑电解码**
- **第三节: 表征相似性分析**
- **第四节: 反向编码模型**

下载并导入需要用到的Python包

In [1]:
```
! pip install inverted_encoding

import numpy as np
import sys
import os
from six.moves import urllib
import gdown
import zipfile
import scipy.io as sio
from scipy.stats import ttest_ind
import h5py
from tqdm import tqdm
from neurora.decoding import tbyt_decoding_kfold, ct_decoding_kfold
from neurora.rsa_plot import plot_tbyt_decoding_acc, \
                             plot_ct_decoding_acc, \
                             plot_rdm, plot_tbytsim_withstats
from neurora.rdm_cal import eegRDM
from neurora.corr_cal_by_rdm import rdms_corr
from neurora.stuff import smooth_1d
from inverted_encoding import IEM, permutation, circ_diff
from decimal import Decimal
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: inverted_encoding in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (0.2.3)
Requirement already satisfied: numpy in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
inverted_encoding) (1.23.5)
```

第一节 - 批处理读取与存储数据

预处理过的Demo数据2：

（你将会在第二节、第三节和第四节中使用） 原数据集基于Bae&Luck2019年发表在Journal of Neuroscience上的文章《Dissociable Decoding of Spatial Attention and Working Memory from EEG Oscillations and Sustained Potentials》实验2的数据
这是一个视觉工作记忆任务，被试要求记忆一个水滴形状的朝向，刺激呈现200ms
经过1300毫秒的延迟后呈现一个随机朝向的水滴形状，被试需要转动鼠标使朝向尽可能和记忆朝向一致
刺激可能有16种不同的朝向以及16种不同的位置
这里仅提取前5个被试的数据，数据为做过预处理并分好段之后的ERP数据（预处理参数见论文原文）并带有每一个试次的朝向和位置标签

In [2]:
```
# 下载Demo数据2

data_dir = "data/"
if not os.path.exists(data_dir):
  os.makedirs(data_dir)

# 从Google Drive下载
url = "https://drive.google.com/file/d/1POBi0dckB00AKCIvpXHZFI1Oqd-jWRoj/view?usp=sharing"
filename = "demo_data2"
filepath = data_dir + filename + ".zip"

# 下载数据
gdown.download(url=url, output=filepath, quiet=False, fuzzy=True)
print("Download completes!")
```

```
# 解压数据
with zipfile.ZipFile(filepath, 'r') as zip:
    zip.extractall(data_dir)
print("Unzip completes!")

# 也可以通过百度网盘下载
# 链接:https://pan.baidu.com/s/1mZdU9rL8vGn8-kjlUkF6PA  密码:8qih
# 下载后解压，并移动到data文件夹下
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1POBi0dckB00AKCIvpXHZFI1Oqd-jWRoj
From (redirected): https://drive.google.com/uc?id=1POBi0dckB00AKCIvpXHZFI1Oqd-jWRoj&confirm=t&uuid=430b6aca-81e1-44b5-a6f4-a2daf145dc2c
To: /Users/zitonglu/Downloads/Python-EEG-Handbook-master/data/demo_data2.zip
100%|| 301M/301M [00:57<00:00, 5.27MB/s]
Download completes!
Unzip completes!
```

在'data'文件夹下包含5个被试的脑电数据.mat文件，在'labels'文件夹下包含每个被试所有试次对应刺激项的朝向标签文件和位置标签文件 以被试编号'201'为例，'ori_201.txt'文件包含其每一个试次的记忆项的朝向信息，'pos_201.txt'文件包含其每一个试次的记忆项的位置信息 数据中，导联数为27，采样率为250Hz，从刺激呈现前1.5s到刺激呈现后1.5s，每个试次包含750个时间点

批处理读取Demo数据2并存成.h5文件

In [3]:
```
# 5个被试的编号
sub_ids = ['201', '202', '203', '204', '205']

# 对5个被试的数据进行遍历
for i, sub in enumerate(sub_ids):

    # 每个被试ERP数据、记忆项朝向标签和记忆项位置标签的文件地址
    subdata_path = 'data/demo_data2/data/ERP' + sub + '.mat'
    suborilabels_path = 'data/demo_data2/labels/ori_' + sub + '.txt'
    subposlabels_path = 'data/demo_data2/labels/pos_' + sub + '.txt'

    # 读取ERP数据
    subdata = sio.loadmat(subdata_path)['filtData']
    # 读取记忆项的朝向和位置的标签
    # 在.txt文件里，第一列为具体的朝向/位置值，第二列为16个朝向/位置对应的标签值（0-15的整数表示）
    suborilabels = np.loadtxt(suborilabels_path)[:, 1]
    subposlabels = np.loadtxt(subposlabels_path)[:, 1]

    # 打印被试编号、ERP数据矩阵的shape及两个标签矩阵的shape
    print('sub' + sub)
    print(subdata.shape)
    print(suborilabels.shape)
    print(subposlabels.shape)

    # 将每个被试的ERP数据及朝向和位置的标签以‘data’、'orilabels'和‘poslabels’
    # 作为Keys存在一个.h5文件里
    f = h5py.File('data/demo_data2/sub' + sub + '.h5', 'w')
    f.create_dataset('data', data=subdata)
    f.create_dataset('orilabels', data=suborilabels)
    f.create_dataset('poslabels', data=subposlabels)
    f.close()
```

```
sub201
(640, 27, 750)
(640,)
(640,)
sub202
(640, 27, 750)
(640,)
(640,)
sub203
(640, 27, 750)
(640,)
(640,)
sub204
(640, 27, 750)
(640,)
(640,)
sub205
(640, 27, 750)
(640,)
(640,)
```

第二节 - 基于分类的脑电解码

Get EEG Data and Labels

In [4]:
```
subs = ["201", "202", "203", "204", "205"]

# 初始化三个变量data、label_ori和label_pos
# 分别用于后续存储脑电数据、朝向的标签和位置的标签
data = np.zeros([5, 640, 27, 500])
label_ori = np.zeros([5, 640])
label_pos = np.zeros([5, 640])

for i, sub in enumerate(subs):

    # 读取单个被试的.h5文件
    subfile = h5py.File('data/demo_data2/sub' + sub + '.h5', 'r')

    # 获取EEG数据
    subdata = subfile['data']

    # subdata的shape为[640, 27, 750]
    # 640 - 试次数； 27 - 导联数； 750 - 时间点数（从-1.5s到1.5s）

    # 取-0.5s到1.5s的数据
    subdata = subdata[:, :, 250:]

    # 读取朝向和位置标签
    sublabel_ori = subfile['orilabels']
    sublabel_pos = subfile['poslabels']

    data[i] = subdata
    label_ori[i] = sublabel_ori
    label_pos[i] = sublabel_pos
```

逐时间点解码与结果可视化

对朝向信息进行逐时间点解码

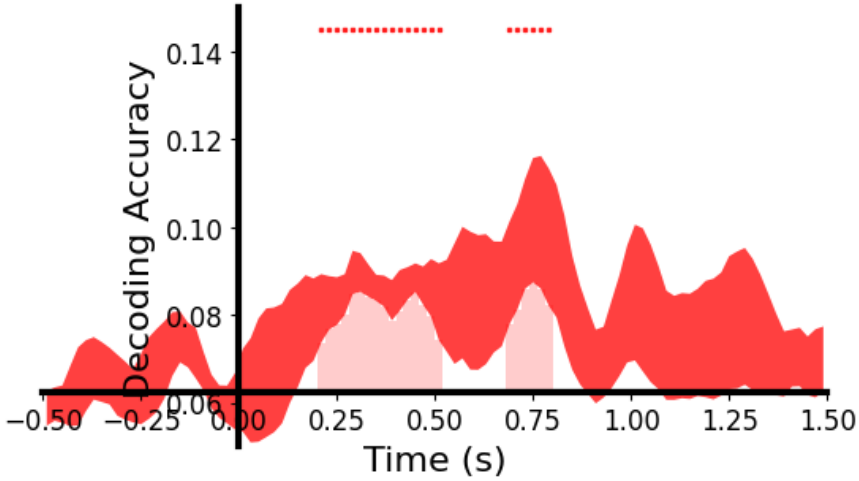
In [5]:
```
# 这里使用NeuroRA的decoding模块下的tbyt_decoding_kfold()函数进行逐时间点解码
# 传入数据（shape为[n_subs, n_trials, n_channels, n_times]）、对应每个试次的标签
# 以及一些其他参数：
# n=16 - 分类类别数为16（共16中朝向）
# navg=13 - 对13个试次进行一次平均
# time_win=5 - 时间窗为5（即5个时间点的数据进行一次平均）
# time_step=5 - 时间步长为5（即每间隔5个时间点进行一次解码）
# nfolds=3 & nrepeats=10 - 进行10次3折的交叉验证
# smooth=True - 对结果进行平滑处理
accs_ori = tbyt_decoding_kfold(data, label_ori, n=16, navg=13, time_win=5, time_step=5,
                               nfolds=3, nrepeats=10, smooth=True)
```

```
Calculating: [================================================================================================================] 100.00%
Decoding finished!
```

对朝向信息解码结果的可视化

In [6]:
```
# 这里使用NeuroRA的decoding模块下的plot_tbyt_decoding_acc()函数进行统计分析与可视化
# 其要求输入解码的结果（shape为[n_subs, n_results_times]）和一些其他参数：
# start_time=-0.5 - 解码结果的开始时间为-0.5s
# end_time=1.5 - 解码结果的结束时间为1.5s、
# time_interval=0.02 - 两解码时间间隔为0.02s
# （由于时间窗为5，数据原采样率为250Hz，5*(1/250)=0.02s）、
# chance=0.0625 - 随机水平的分类准确率为0.0625（16个朝向，即1/16）
# p=0.05 - p值的阈值、
# cbpt=True - 进行cluster-based permutation test、
# stats_time=[0, 1.5] - 统计0-1.5s的解码结果、
# xlim=[-0.5, 1.5] - x轴显示范围从-0.5到1.5
# ylim=[0.05, 0.15] - y轴显示范围从0到1.5
plot_tbyt_decoding_acc(accs_ori, start_time=-0.5, end_time=1.5, time_interval=0.02,
                       chance=0.0625, p=0.05, cbpt=True, stats_time=[0, 1.5],
                       xlim=[-0.5, 1.5], ylim=[0.05, 0.15])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!


Significant time-windows:
200ms to 520ms
679ms to 800ms
```



Out[6]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

对位置信息进行逐时间点解码

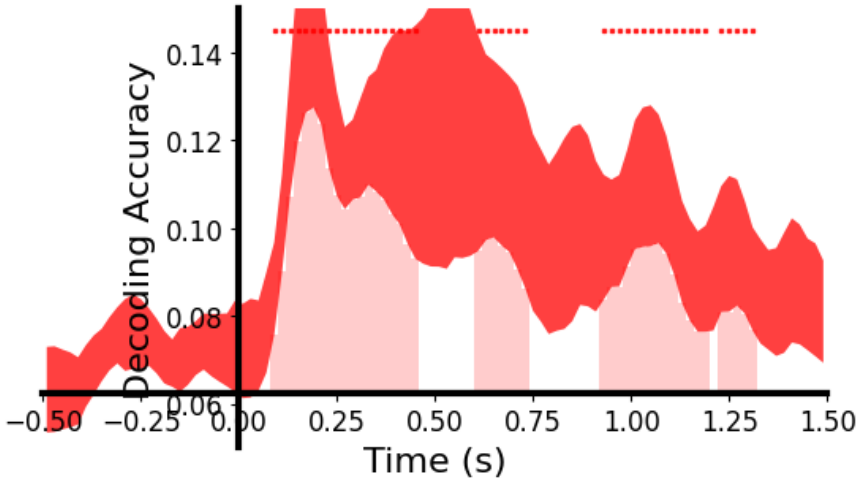
In [7]:
```
accs_pos = tbyt_decoding_kfold(data, label_pos, n=16, navg=13, time_win=5, time_step=5,
                               nfolds=3, nrepeats=10, smooth=True)
```

```
Calculating: [================================================================================
==============] 100.00%
Decoding finished!
```

对位置信息解码结果的可视化

In [8]:
```
plot_tbyt_decoding_acc(accs_pos, start_time=-0.5, end_time=1.5, time_interval=0.02,
                       chance=0.0625, p=0.05, cbpt=True, stats_time=[0, 1.5],
                       xlim=[-0.5, 1.5], ylim=[0.05, 0.15])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!


Significant time-windows:
79ms to 459ms
600ms to 740ms
919ms to 1200ms
1220ms to 1320ms
```



Out[8]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

跨时域脑电解码

对朝向和位置信息进行跨时域脑电解码

朝向解码

In [9]:
```
# 这里使用NeuroRA的decoding模块下的ct_decoding_kfold()函数进行跨时域解码
accs_crosstime_ori = ct_decoding_kfold(data, label_ori, n=16, navg=13,
                                       time_win=5, time_step=5,
                                       nfolds=3, nrepeats=10, smooth=True)
```

```
Decoding
Calculating: [================================================================================
==============] 100.00%
Decoding finished!
```

位置解码

In [10]:
```
accs_crosstime_pos = ct_decoding_kfold(data, label_pos, n=16, navg=13,
                                       time_win=5, time_step=5,
                                       nfolds=3, nrepeats=10, smooth=True)
```

```
Decoding
Calculating: [================================================================================
==============] 100.00%
Decoding finished!
```

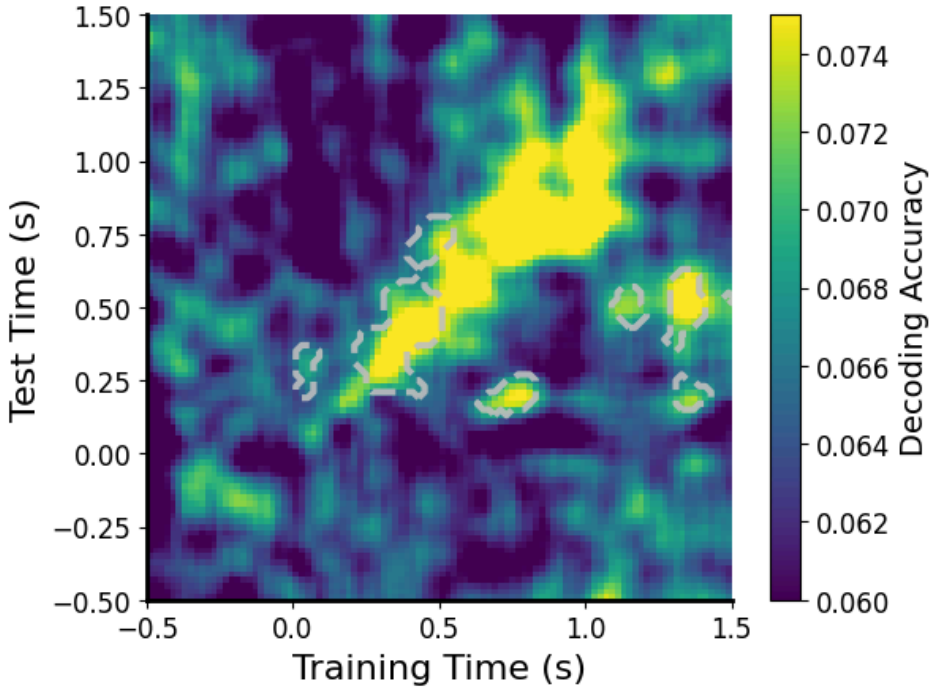
对朝向和位置跨时域解码结果的可视化

朝向解码

In [11]:
```
# 这里使用NeuroRA的decoding模块下的plot_ct_decoding_acc()函数进行统计分析与可视化
plot_ct_decoding_acc(accs_crosstime_ori, start_timex=-0.5, end_timex=1.5,
                     start_timey=-0.5, end_timey=1.5,
                     time_intervalx=0.02, time_intervaly=0.02,
                     chance=0.0625, p=0.05, cbpt=True,
                     stats_timex=[0, 1.5], stats_timey=[0, 1.5],
                     xlim=[-0.5, 1.5], ylim=[-0.5, 1.5], clim=[0.06, 0.075])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!
```

```
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/neurora/rsa_plot.py:1050: UserWarning: No con
tour levels were found within the data range.
  plt.contour(X, Y, np.transpose(newps, (1, 0)), [0, 1], colors="silver", alpha=0.9, linewidths=3,
```

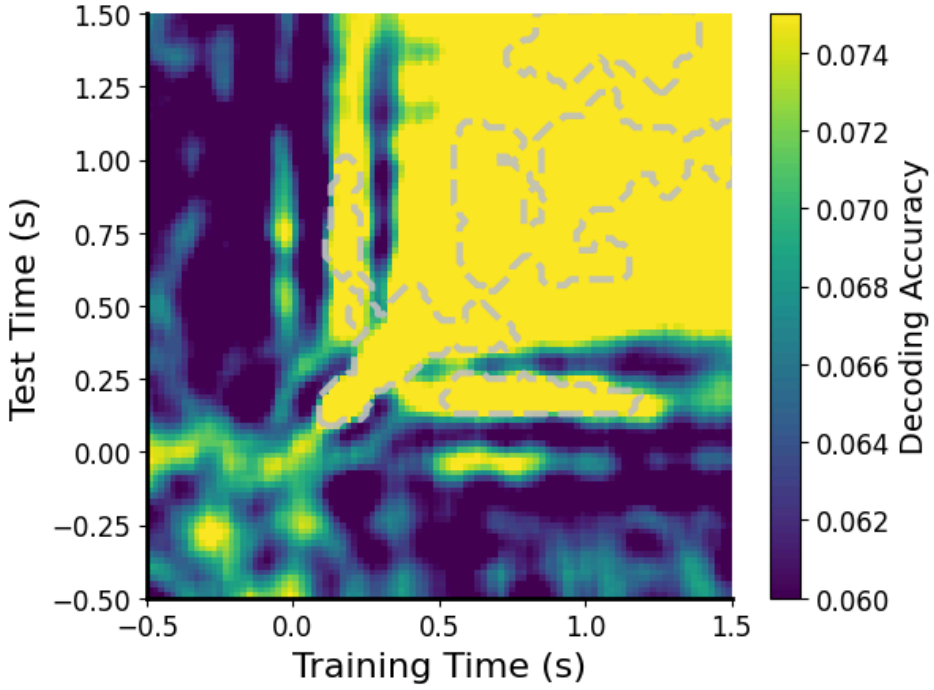


Out[11]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

位置解码

In [12]:
```
plot_ct_decoding_acc(accs_crosstime_pos, start_timex=-0.5, end_timex=1.5,
                     start_timey=-0.5, end_timey=1.5,
                     time_intervalx=0.02, time_intervaly=0.02,
                     chance=0.0625, p=0.05, cbpt=True,
                     stats_timex=[0, 1.5], stats_timey=[0, 1.5],
                     xlim=[-0.5, 1.5], ylim=[-0.5, 1.5], clim=[0.06, 0.075])
```

```
Permutation test
Calculating: [============================================================================================================] 100.00%
Cluster-based permutation test finished!
```



Out[12]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

第三节 - 表征相似性分析

计算脑电表征不相似性矩阵（Representational Dissimilarity Matrices，RDMs）

根据16个朝向和16个位置，分别计算逐时间点的16×16的表征朝向信息的脑电RDM和表征位置信息的脑电RDM

In [13]:
```python
n_subs = 5
n_trials = 640
# 首先对数据进行重新划分
# 分别获取16个朝向条件下的脑电数据和16个位置条件下的脑电数据
# 初始化两个变量data_ori和data_pos分别用于存储朝向和位置的数据
data_ori = np.zeros([16, n_subs, 40, 27, 500])
data_pos = np.zeros([16, n_subs, 40, 27, 500])
for sub in range(n_subs):
    index_ori = np.zeros([16], dtype=int)
    index_pos = np.zeros([16], dtype=int)
    for i in range(n_trials):
        ori = int(label_ori[sub, i])
        pos = int(label_pos[sub, i])
        data_ori[ori, sub, index_ori[ori]] = data[sub, i]
        index_ori[ori] = index_ori[ori] + 1
        data_pos[pos, sub, index_pos[pos]] = data[sub, i]
        index_pos[pos] = index_pos[pos] + 1

# 使用NeuroRA的rdm_cal模块下的eegRDM()函数计算脑电RDMs
# 朝向RDMs
RDMs_ori = eegRDM(data_ori, sub_opt=1, chl_opt=0, time_opt=1, time_win=5, time_step=5)
# 位置RDMs
RDMs_pos = eegRDM(data_pos, sub_opt=1, chl_opt=0, time_opt=1, time_win=5, time_step=5)
# 返回的RDMs_ori和RDMs_pos的shape均为[n_subs即5, n_results_time即100, 16, 16]
```

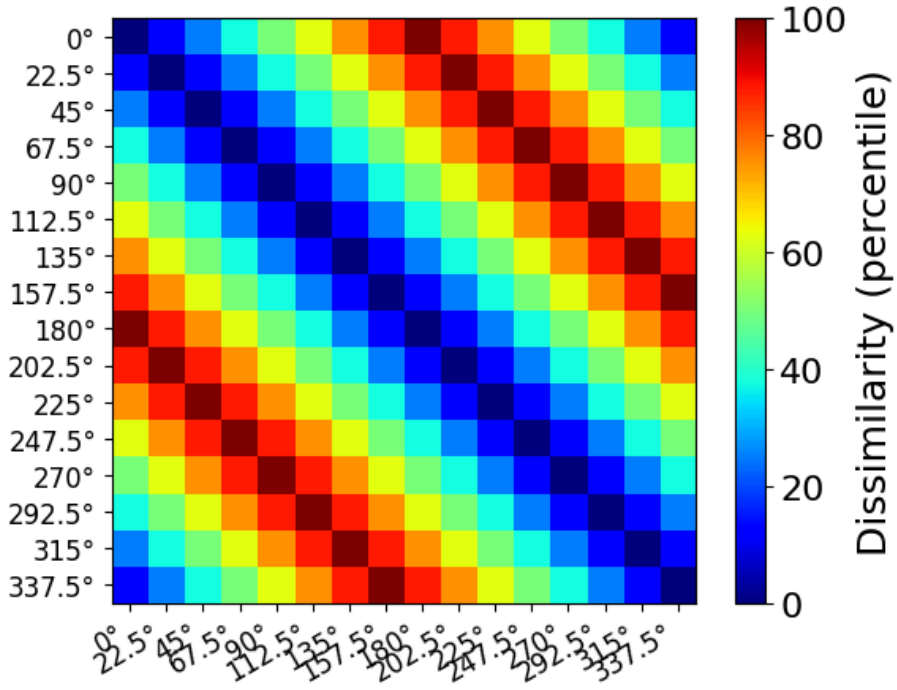
```
Computing RDMs
Calculating: [================================================================================
==============] 100.00%
RDMs computing finished!

Computing RDMs
Calculating: [================================================================================
==============] 100.00%
RDMs computing finished!
```

构建基于假设的表征不相似矩阵

这里基于两角度越接近相似性约高、两角度越不接近相似性越低的假设
构建了一个编码模型RDM，此例中，对朝向和位置信息均适用

In [14]:
```python
model_RDM = np.zeros([16, 16])
for i in range(16):
    for j in range(16):
        diff = np.abs(i - j)
        if diff <= 8:
            model_RDM[i, j] = diff / 8
        else:
            model_RDM[i, j] = (16 - diff) / 8

# 使用NeuroRA的rsa_plot模块下的plot_rdm()函数可视化RDM
conditions = ["0°", "22.5°", "45°", "67.5°", "90°", "112.5°", "135°", "157.5°", "180°",
              "202.5°", "225°", "247.5°", "270°", "292.5°", "315°", "337.5°"]
plot_rdm(model_RDM, percentile=True, conditions=conditions)
```

```
1
```



Out[14]: 0

相似性分析（Representational Similarity Analysis, RSA）

使用基于假设的模型RDM分别与朝向表征的脑电RDMs和位置表征的脑电RDMs计算相似性
来时序上追踪大脑何时符合对朝向的编码、何时符合对位置的编码

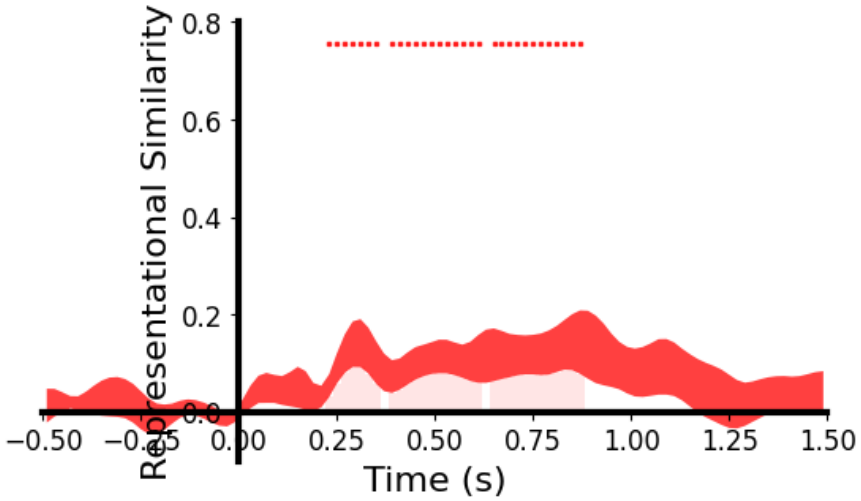
In [15]:
```
# 使用NeuroRA的corr_cal_by_rdm模块下的rdms_corr()函数
# 计算基于假设的RDM和朝向的EEG RDMs的相似性
similarities_ori = rdms_corr(model_RDM, RDMs_ori)
# 计算基于假设的RDM和位置的EEG RDMs的相似性
similarities_pos = rdms_corr(model_RDM, RDMs_pos)
```

```
Computing similarities

Computing finished!

Computing similarities

Computing finished!
```

基于RSA的大脑对朝向信息编码的结果可视化

In [16]:
```
# 使用NeuroRA的rsa_plot模块下的plot_tbytsim_withstats()函数
# 用法与plot_tbyt_decoding_acc()函数类似
plot_tbytsim_withstats(similarities_ori, start_time=-0.5, end_time=1.5,
                       time_interval=0.02, smooth=True, p=0.05, cbpt=True,
                       stats_time=[0, 1.5], xlim=[-0.5, 1.5], ylim=[-0.1, 0.8])
```

```
Permutation test
Calculating: [================================================================================================
==============] 100.00%
Cluster-based permutation test finished!


Significant time-windows:
219ms to 360ms
380ms to 620ms
640ms to 880ms
```

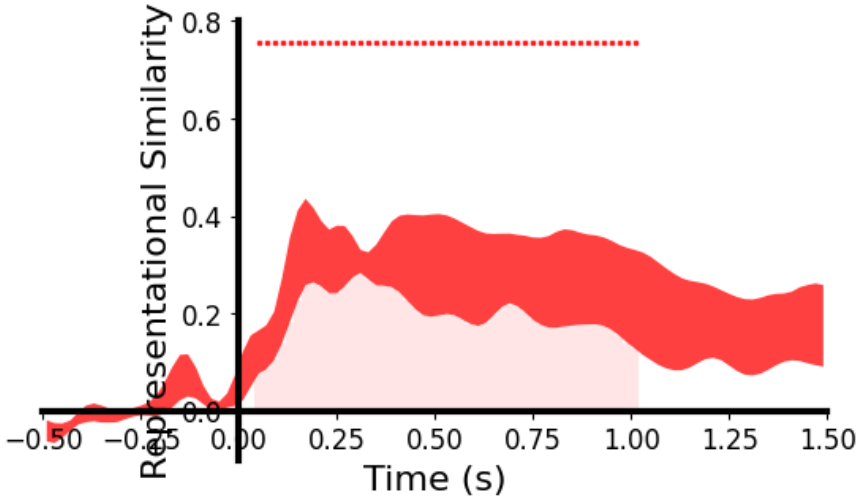


Out[16]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

基于RSA的大脑对位置信息编码的结果可视化

In [17]:
```
plot_tbytsim_withstats(similarities_pos, start_time=-0.5, end_time=1.5,
                       time_interval=0.02, smooth=True, p=0.05, cbpt=True,
                       stats_time=[0, 1.5], xlim=[-0.5, 1.5], ylim=[-0.1, 0.8])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!


Significant time-windows:
40ms to 1020ms
```



Out[17]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

第四节 - 反向编码模型

这部分，我们参考预印本"*Scotti, P. S., Chen, J., & Golomb, J. D. (2021). An enhanced inverted encoding model for neural reconstructions. bioRxiv*"使用强化版反向编码模型（enhanced inverted encoding model, eIEM）

使用eIEM解码朝向信息

In [18]:
```
n_subs = 5
n_ts = 100
```

```
# 初始化mae_ori来存储IEM解码的MAE结果
mae_ori = np.zeros([n_subs, n_ts])

# 使用Inverted_Encoding的IEM函数计算逐被试、逐时间点的MAE
for t in tqdm(range(n_ts)):
    for sub in range(n_subs):

        # 降采样数据 - 即对每5个时间点平均
        data_t_sub = np.average(data[sub, :, :, t*5:t*5+5], axis=2)

        # 获取IEM预测结果
        predictions, _, _, _ = IEM(data_t_sub, label_ori[sub].astype(int),
                                   stim_max=16, nfolds=5, is_circular=True)

        # 计算MAE
        mae_ori[sub, t] = np.mean(np.abs(circ_diff(predictions, label_ori[sub].astype(int), 16)))

# 使用Inverted_Encoding的permutation()函数获取MAE的Null分布
null_mae_ori = permutation(label_ori[sub].astype(int), stim_max=16,num_perm=5000, is_circular=True)
```

```
100%|██████████████████████████████████████| 100/100 [01:28<00:00,  1.13it/s]
```

定义绘制eIEM结果的函数 - plot_iem_results()

为了可视化，我们计算ΔMAE（null MAE的均值减MAE）作为最终的IEM结果

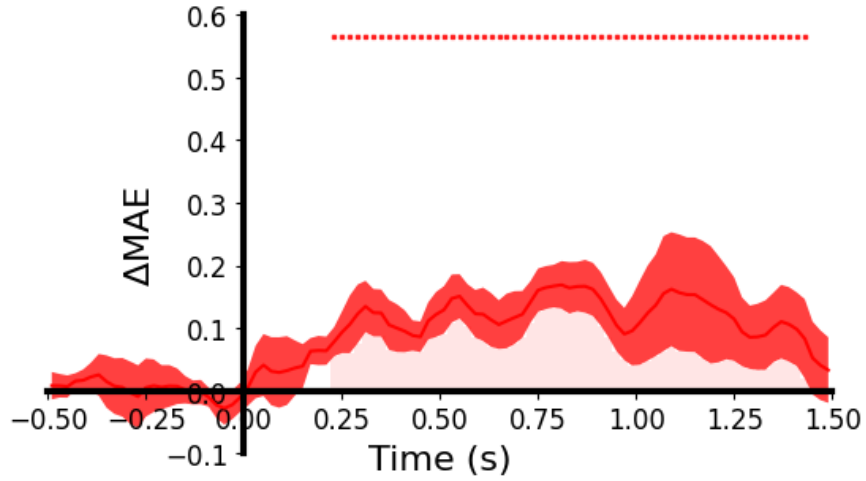
In [19]:
```
def plot_iem_results(mae_ori, null_mae, start_time=0, end_time=1, time_interval=0.01,
                     stats_time=[0, 1], xlim=[0, 1], ylim=[-0.1, 0.8]):

    """
    参数:
        mae: shape为[n_subs, n_times]的矩阵，对应每个被试的MAE结果
        null_mae: shape为[num_perm]的矩阵，对应Null分布
        start_time: 起始时间，默认为0
        end_time: 结束时间，默认为1
        time_interval: 两时间点之间的间隔，默认为0.01
        xlim: X轴范围，默认[0, 1]
        ylim: Y轴范围，默认[0.4, 0.8]
    """

    if len(np.shape(mae_ori)) != 2:

        return "Invalid input!"

    n = len(np.shape(mae_ori))

    yminlim = ylim[0]
    ymaxlim = ylim[1]

    nsubs, nts = np.shape(mae_ori)
    tstep = float(Decimal((end_time - start_time) / nts).quantize(Decimal(str(time_interval))))

    if tstep != time_interval:
        return "Invalid input!"

    delta1 = (stats_time[0] - start_time) / tstep - int((stats_time[0] - start_time) / tstep)
    delta2 = (stats_time[1] - start_time) / tstep - int((stats_time[1] - start_time) / tstep)

    mae_ori = smooth_1d(mae_ori)

    avg = np.average(np.average(null_mae)-mae_ori, axis=0)
    err = np.zeros([nts])

    for t in range(nts):
        err[t] = np.std(mae_ori[:, t], ddof=1)/np.sqrt(nsubs)

    ps = np.zeros([nts])
    for t in range(nts):
        ps[t] = ttest_ind(mae_ori[:, t], null_mae, alternative='less')[1]
        if ps[t] < 0.05:
            ps[t] = 1
        else:
            ps[t] = 0

    print('\nSignificant time-windows:')
    for t in range(nts):
        if t == 0 and ps[t] == 1:
            print(str(int(start_time * 1000)) + 'ms to ', end='')
        if t > 0 and ps[t] == 1 and ps[t - 1] == 0:
```

```
            print(str(int((start_time + t * tstep) * 1000)) + 'ms to ', end='')
        if t < nts - 1 and ps[t] == 1 and ps[t + 1] == 0:
            print(str(int((start_time + (t + 1) * tstep) * 1000)) + 'ms')
        if t == nts - 1 and ps[t] == 1:
            print(str(int(end_time * 1000)) + 'ms')

    for t in range(nts):
        if ps[t] == 1:
            plt.plot(t*tstep+start_time+0.5*tstep, (ymaxlim-yminlim)*0.95+yminlim, 's',
                     color='r', alpha=0.8, markersize=2)
            xi = [t*tstep+start_time, t*tstep+tstep+start_time]
            ymin = [0]
            ymax = [avg[t]-err[t]]
            plt.fill_between(xi, ymax, ymin, facecolor='r', alpha=0.1)

    fig = plt.gcf()
    fig.set_size_inches(6.4, 3.6)

    ax = plt.gca()
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_linewidth(3)
    ax.spines["left"].set_position(("data", 0))
    ax.spines["bottom"].set_linewidth(3)
    ax.spines['bottom'].set_position(('data', 0))
    x = np.arange(start_time+0.5*tstep, end_time+0.5*tstep, tstep)
    plt.plot(x, avg, color='r', alpha=0.95)
    plt.fill_between(x, avg + err, avg - err, facecolor='r', alpha=0.75)
    plt.ylim(yminlim, ymaxlim)
    plt.xlim(xlim[0], xlim[1])
    plt.tick_params(labelsize=12)
    plt.xlabel('Time (s)', fontsize=16)
    plt.ylabel(r'$\Delta$MAE', fontsize=16)

    plt.show()

    return ps
```

绘制朝向信息的IEM结果

In [20]:
```
plot_iem_results(mae_ori, null_mae_ori, start_time=-0.5, end_time=1.5, time_interval=0.02,
                 xlim=[-0.5, 1.5], ylim=[-0.1, 0.6])
```

```
Significant time-windows:
219ms to 1440ms
```



Out[20]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0.])
```

使用eIEM解码位置信息

In [21]:
```
# 初始化mae_pos来存储IEM解码的MAE结果
mae_pos = np.zeros([n_subs, n_ts])

# 使用Inverted_Encoding的IEM函数计算逐被试、逐时间点的MAE
for t in tqdm(range(n_ts)):
    for sub in range(n_subs):
```

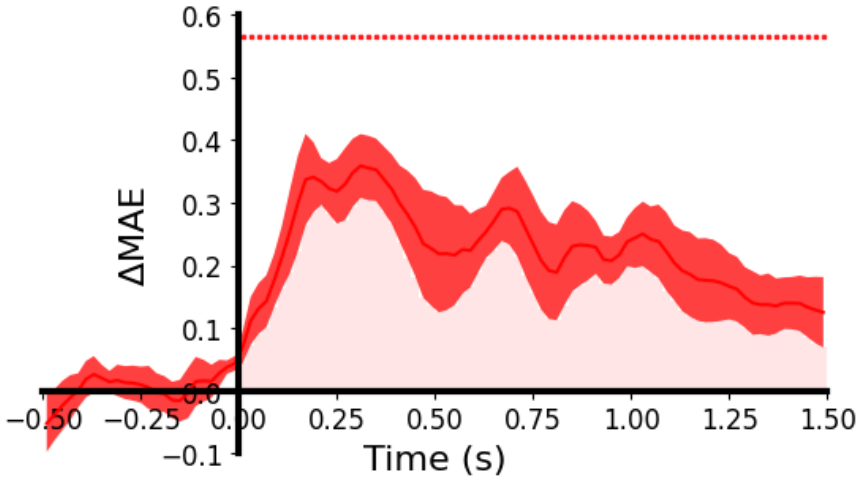
```
        # 降采样数据 - 即对每5个时间点平均
        data_t_sub = np.average(data[sub, :, :, t*5:t*5+5], axis=2)

        # 获取IEM预测结果
        predictions, _, _, _ = IEM(data_t_sub, label_pos[sub].astype(int), stim_max=16, nfolds=5, is

        # 计算MAE
        mae_pos[sub, t] = np.mean(np.abs(circ_diff(predictions, label_pos[sub].astype(int), 16)))

# 使用Inverted_Encoding的permutation()函数获取MAE的Null分布
null_mae_pos = permutation(label_pos[sub].astype(int), stim_max=16, num_perm=5000, is_circular=True)
```

```
100%|██████████████████████████████████████| 100/100 [01:28<00:00,  1.13it/s]
```

绘制位置信息的IEM结果

In [22]:
```
plot_iem_results(mae_pos, null_mae_pos, start_time=-0.5, end_time=1.5, time_interval=0.02,
                 xlim=[-0.5, 1.5], ylim=[-0.1, 0.6])
```

```
Significant time-windows:
0ms to 1500ms
```



Out[22]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

关于此手册

源码获取：

见 GitHub 项目仓库：
中文版：https://github.com/ZitongLu1996/Python-EEG-Handbook-CN
英文版：https://github.com/ZitongLu1996/Python-EEG-Handbook

文档获取：

见 GitHub 项目仓库：
中文版：https://github.com/ZitongLu1996/Python-EEG-Handbook-CN
英文版：https://github.com/ZitongLu1996/Python-EEG-Handbook

问题与反馈：
如若对本手册有任何疑问或建议，欢迎通过以下方式联系：
（1） 在公众号“路同学”后台留言；
（2） GitHub 项目仓库 Issues 板块留言；
（3） 邮件联系路子童，邮箱：zitonglu1996@gmail.com

引用：

Lu, Z., Li, W., Nie, L., & Zhao, K. (2024) An Easy-to-Follow Handbook for EEG Data Analysis based on Python. PsyArXiv.
https://doi.org/10.31234/osf.io/dcmke

致谢

在编写过程中，感谢微信群“MNE-Python 学习交流群”和“NeuroRA 纳谏答疑群”群内成员以及路同学公众号关注者们的积极支持与反馈，感谢贾会宾老师的《EEGLAB 中文手册》、Steven Luck 的《事件相关电位基础》、EEGLAB 与 MNE-Python 的官网文档与教程给我们提供了宝贵的参考，感谢 OSU 的本科生冉孟馨对第一版文档进行的审阅，也要感谢对第一版推送进行转载的其他公众号的运营者以及所有在初版文档发布后的帮助积极宣传与分享的朋友们。衷心感谢在这个过程中每一个提供种种帮助的人！