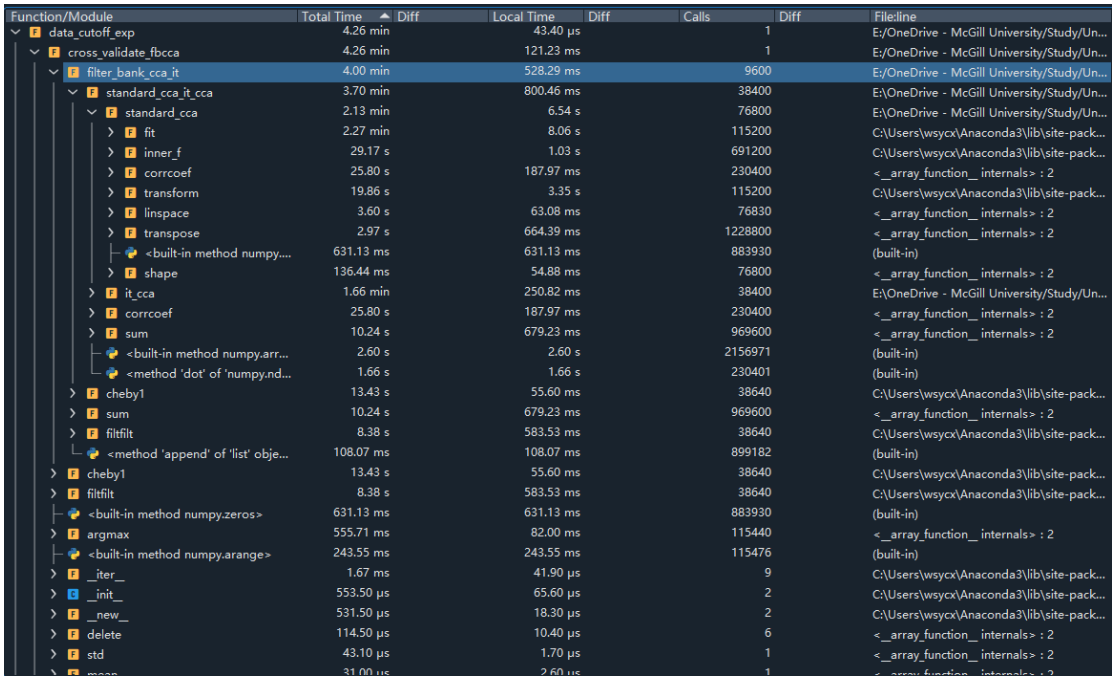
CCA Run Time Analysis

By Mike Wang

Part I. Python-based CCA (Based on ***main_mike.py***)



Function/Module	Total Time	Diff	Local Time	Diff	Calls	Diff	File/line
data_cutoff_exp	4.26 min		43.40 µs		1		E:/OneDrive - McGill University/Study/Un...
cross_validate_fbcca	4.26 min		121.23 ms		1		E:/OneDrive - McGill University/Study/Un...
filter_bank_cca_it	4.00 min		528.29 ms		9600		E:/OneDrive - McGill University/Study/Un...
standard_cca_it_cca	3.70 min		800.46 ms		38400		E:\OneDrive - McGill University/Study/Un...
standard_cca	2.13 min		6.54 s		76800		E:\OneDrive - McGill University/Study/Un...
fit	2.27 min		8.06 s		115200		C:\Users\wsycx\Anaconda3\lib\site-pack...
inner_f	29.17 s		1.03 s		691200		C:\Users\wsycx\Anaconda3\lib\site-pack...
corrcoef	25.80 s		187.97 ms		230400		<__array_function__ internals> : 2
transform	19.86 s		3.35 s		115200		C:\Users\wsycx\Anaconda3\lib\site-pack...
linspace	3.60 s		63.08 ms		76830		<__array_function__ internals> : 2
transpose	2.97 s		664.39 ms		1228800		<__array_function__ internals> : 2
<built-in method numpy....	631.13 ms		631.13 ms		883930		(built-in)
shape	136.44 ms		54.88 ms		76800		<__array_function__ internals> : 2
it_cca	1.66 min		250.82 ms		38400		E:\OneDrive - McGill University/Study/Un...
corrcoef	25.80 s		187.97 ms		230400		<__array_function__ internals> : 2
sum	10.24 s		679.23 ms		969600		<__array_function__ internals> : 2
<built-in method numpy.arr...	2.60 s		2.60 s		2156971		(built-in)
<method 'dot' of 'numpy.nd...	1.66 s		1.66 s		230401		(built-in)
cheby1	13.43 s		55.60 ms		38640		C:\Users\wsycx\Anaconda3\lib\site-pack...
sum	10.24 s		679.23 ms		969600		<__array_function__ internals> : 2
filtfilt	8.38 s		583.53 ms		38640		C:\Users\wsycx\Anaconda3\lib\site-pack...
<method 'append' of 'list' obje...	108.07 ms		108.07 ms		899182		(built-in)
cheby1	13.43 s		55.60 ms		38640		C:\Users\wsycx\Anaconda3\lib\site-pack...
filtfilt	8.38 s		583.53 ms		38640		C:\Users\wsycx\Anaconda3\lib\site-pack...
<built-in method numpy.zeros>	631.13 ms		631.13 ms		883930		(built-in)
argmax	555.71 ms		82.00 ms		115440		<__array_function__ internals> : 2
<built-in method numpy.arange>	243.55 ms		243.55 ms		115476		(built-in)
__iter__	1.67 ms		41.90 µs		9		C:\Users\wsycx\Anaconda3\lib\site-pack...
__init__	553.50 µs		65.60 µs		2		C:\Users\wsycx\Anaconda3\lib\site-pack...
__new__	531.50 µs		18.30 µs		2		C:\Users\wsycx\Anaconda3\lib\site-pack...
delete	114.50 µs		10.40 µs		6		<__array_function__ internals> : 2
std	43.10 µs		1.70 µs		1		<__array_function__ internals> : 2

Figure 1. *Major components time occupation with 1 iteration based on dataset s2. (a complete time profile file has been stored in directory McGill_NeuroTech_Signal\run_time_analysis\python\ main_mike_runtime.Result)*

The major time-consuming functions are the CCA algorithms. One of the reasons why is they have been called for a large amount of time. For example, *filter_bank_cca_it* has been called 9600 times. Therefore, to find out the real time for each function running once, we need to divide their total occupation time by the number of calls. Thus, a detailed run time map for each function has been summarized in Table I.

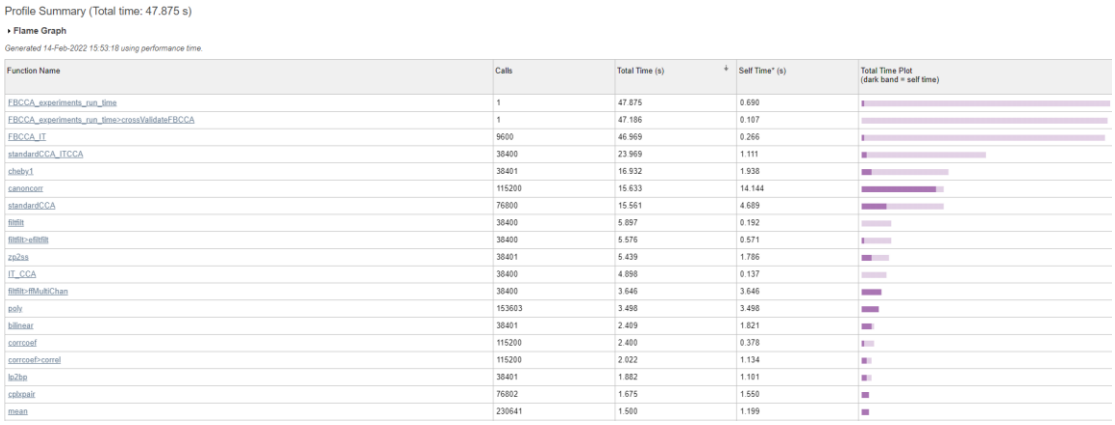
Table I. *Function occupation time for each call*

Function names	**Runtime for each Call (seconds)**
filter_bank_cca_it()	0.025
standard_cca_it_cca()	0.00578125
Standard_cca()	0.0016640625
It_cca()	0.0025937499999999997
Cheby1()	0.0003475672877846791
Filtfilt()	0.0002427536231884058
Corroef()	0.00011197916666666667

Therefore, the time need to run 1 40-fundation-frequency-*filter_bank_cca_it* is **1 second**. Which means in a real word situation, when analyzing 9-channels, 500-samples, 1 target data's filter bank

CCA requires 1 seconds. For a real-world spelling implementation, it is too slow.

Part II. Matlab-based CCA (Based on ***FBCCA_experiments_tun_time.m***)



Profile Summary (Total time: 47.875 s)

▸ Flame Graph

Generated 14-Feb-2022 15:53:18 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
FBCCA_experiments_run_time	1	47.875	0.690	
FBCCA_experiments_run_time>crossValidateFBCCA	1	47.186	0.107	
FBCCA_IT	9600	46.969	0.266	
standardCCA_ITCCA	38400	23.969	1.111	
cheby1	38401	16.932	1.938	
canoncorr	115200	15.633	14.144	
standardCCA	76800	15.561	4.689	
filtfilt	38400	5.897	0.192	
filtfilt>efiltfilt	38400	5.576	0.571	
zp2ss	38401	5.439	1.786	
IT_CCA	38400	4.898	0.137	
filtfilt>fiMultiChan	38400	3.646	3.646	
poly	153603	3.498	3.498	
bilinear	38401	2.409	1.821	
corrcoef	115200	2.400	0.378	
corrcoef>corrcol	115200	2.022	1.134	
lp2bp	38401	1.882	1.101	
cplxpair	76802	1.675	1.550	
mean	230641	1.500	1.199	

Figure 2. *Major components time occupation with 1 iteration based on dataset s2. (a complete time profile file has been stored in directory McGill_NeuroTech_Signal\run_time_analysis\MATLAB\ Matlab-based CCA Run time analysis.pdf)*

A detailed run time map for each function has been summarized in Table II. From above we can tell that MATLAB requires much less time to run a filter bank CCA algorithm.

Table II. *Function occupation time for each call*

Function names	**Runtime for each Call (seconds)**
FBCCA_IT()	0.0049
standardCCA_ITCCA ()	6.2419e-04
Standard_cca()	2.0262e-04
It_cca()	1.2755e-04
Cheby1()	4.4093e-04
Filtfilt()	1.5357e-04
Canoncorr()	1.3570e-04

Therefore, the time need to run 1 40-fundation-frequency-*filter_bank_cca_it* is **0.1960 second**. Which means in a real word situation, when analyzing 9-channels, 500-samples, 1 target data's filter bank CCA requires 0.1960 seconds. For a real-world spelling implementation, it is Ok.