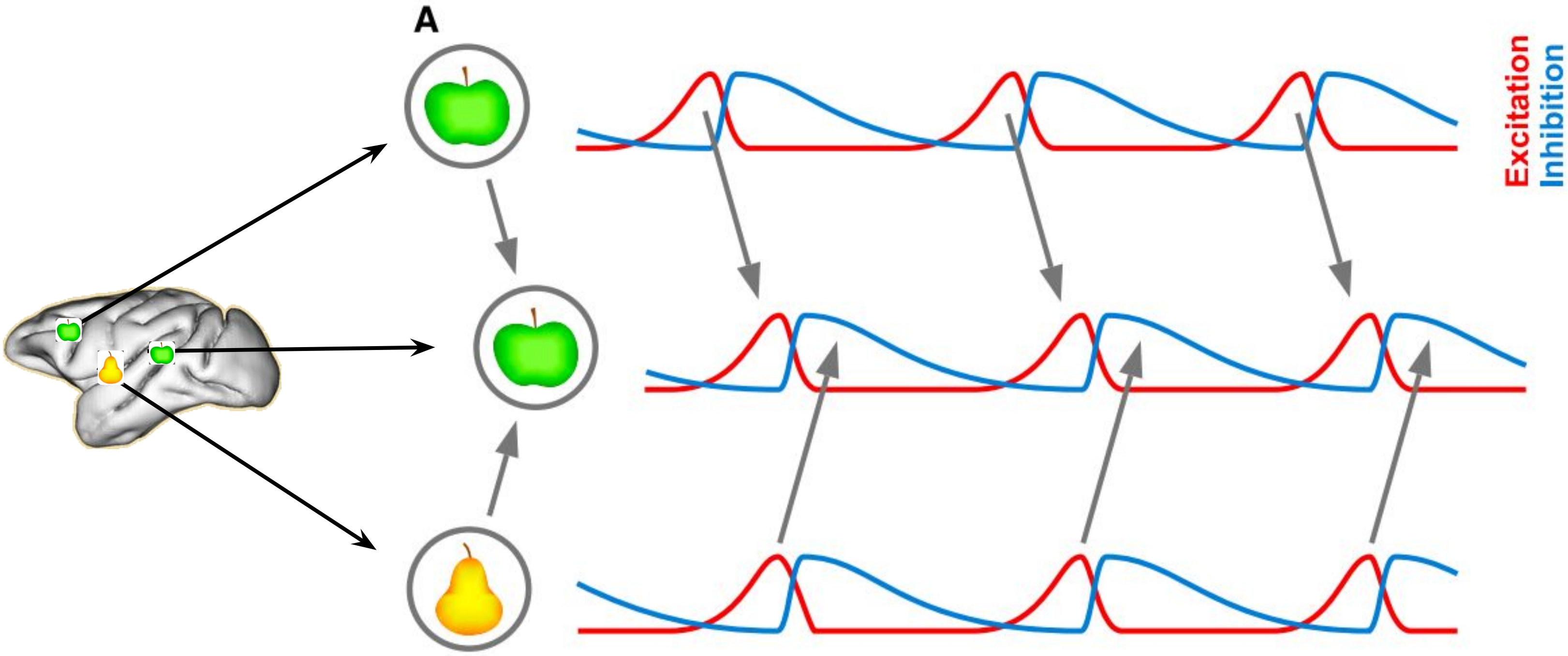
Integration of the single-trial time-resolved spectral connectivity (coherence; PLV) in Frites

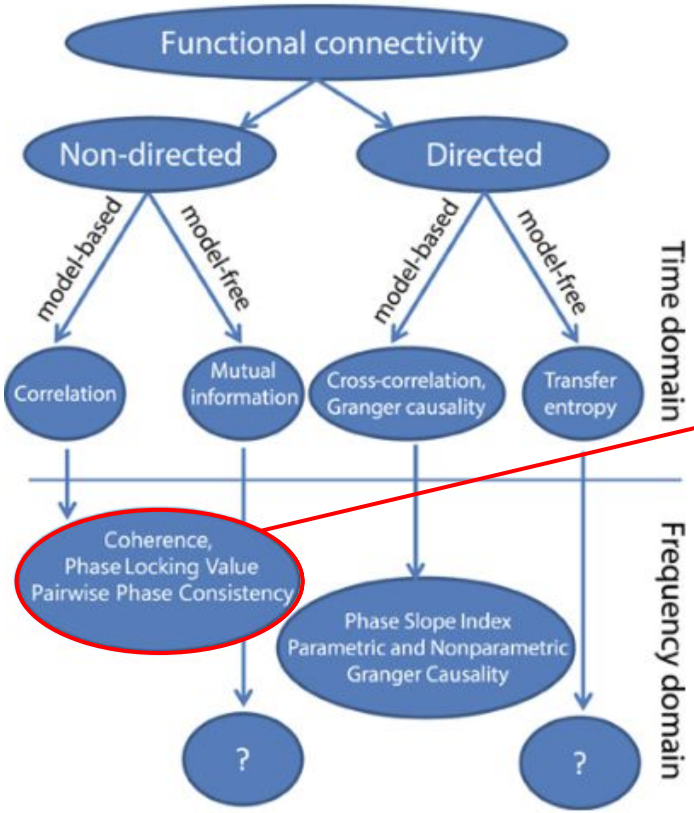
by Vinicius Lima &
Etienne Combrisson

Introduction - communication through coherence



Fries (2015)

Introduction - assessing dFC



If the hypothesis is that sync holds relevant information for, as an example, stimuli encoding. Then the FC can be estimated by measuring the phase coupling (PLV) or phase-amplitude coupling (coherence).

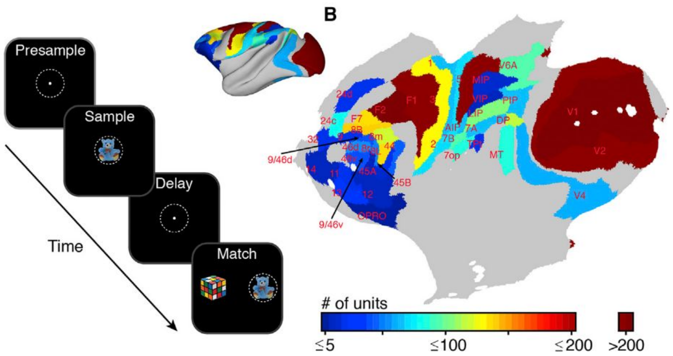
- Values between 0 and 1, for coherence one value per frequency analysed.

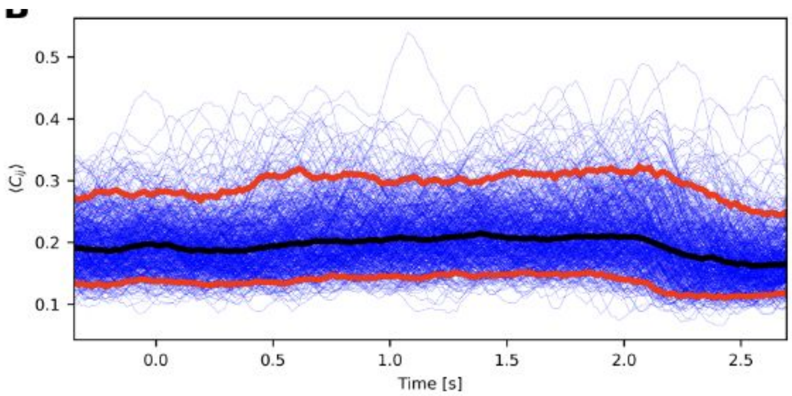
- Classically estimated via Fourier (coh) or Hilbert transforms (PLV);

- Can be extended to time-frequency domain by applying techniques such as windowed Fourier, Wavelet or multitaper analysis;

- Should also be assessed at single-trial level;

Introduction - Why time-resolved and single trial?



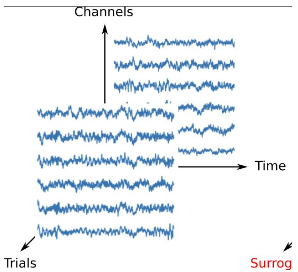


How the dynamics of some quantity related to the registered activity relates with experimental condition.

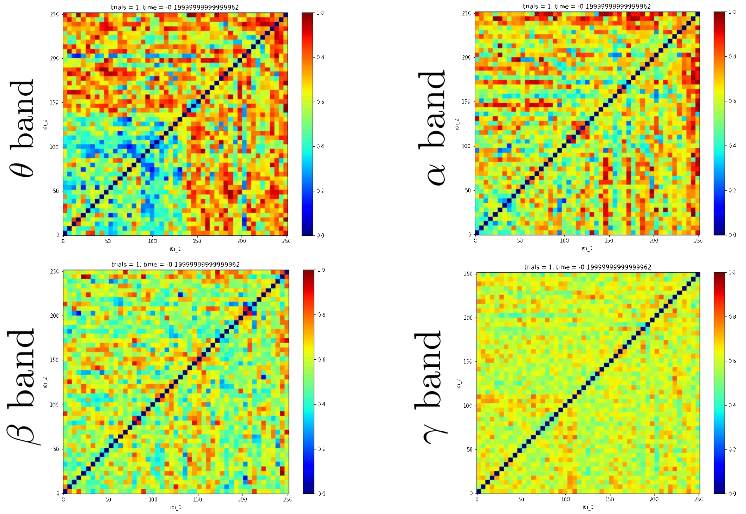
Bursts of coherent activity are likely to not be time-aligned across trials and averaging would destroy this information.

Current state

```python
def _coh(w, kernel, foi_idx, x_s, x_t, kw_para):
    """Pairwise coherence."""
    # auto spectra (faster that w * w.conj())
    s_auto = w.real ** 2 + w.imag ** 2

    # smooth the auto spectra
    s_auto = _smooth_spectra(s_auto, kernel)

    # define the pairwise coherence
    def pairwise_coh(w_x, w_y):
        # computes the coherence
        s_xy = w[:, w_y, :, :] * np.conj(w[:, w_x, :, :])
        s_xy = _smooth_spectra(s_xy, kernel)
        s_xx = s_auto[:, w_x, :, :]
        s_yy = s_auto[:, w_y, :, :]
        out = np.abs(s_xy) ** 2 / (s_xx * s_yy)
        # mean inside frequency sliding window (if needed)
        if isinstance(foi_idx, np.ndarray):
            return _foi_average(out, foi_idx)
        else:
            return out

    # define the function to compute in parallel
    parallel, p_fun = parallel_func(pairwise_coh, **kw_para)

    # compute the single trial coherence
    return parallel(p_fun(s, t) for s, t in zip(x_s, x_t))


def _plv(w, kernel, foi_idx, x_s, x_t, kw_para):
    """Pairwise phase-locking value."""
    # define the pairwise plv
    def pairwise_plv(w_x, w_y):
        # computes the plv
        s_xy = w[:, w_y, :, :] * np.conj(w[:, w_x, :, :])
        # complex exponential of phase differences
        exp_dphi = s_xy / np.abs(s_xy)
        # smooth e^(-i*\delta\phi)
        exp_dphi = _smooth_spectra(exp_dphi, kernel)
        # computes plv
        out = np.abs(exp_dphi)
        # mean inside frequency sliding window (if needed)
        if isinstance(foi_idx, np.ndarray):
            return _foi_average(out, foi_idx)
        else:
            return out

    # define the function to compute in parallel
    parallel, p_fun = parallel_func(pairwise_plv, **kw_para)

    # compute the single trial coherence
    return parallel(p_fun(s, t) for s, t in zip(x_s, x_t))
```




```python
def conn_spec(
        data, freqs=None, metric='coh', roi=None, times=None, pairs=None,
        sfreq=None, foi=None, sm_times=.5, sm_freqs=1, sm_kernel='hanning',
        mode='morlet', n_cycles=7., mt_bandwidth=None, decim=1, kw_cwt={},
        kw_mt={}, block_size=None, n_jobs=-1, verbose=None, dtype=np.float32):
```



Goals

- Describe the methods to other participants;
- Refine the method implementation to estimate spectral connectivity present in xfrites;
- Create the documentation for the method;
- Write smoke and functional unit tests;
- Create examples illustrating the purpose of the single-trial coherence / PLV.