Workflow steps for wavelet – info theory data processing (ProcessNetwork code)

Although the process network code can be run in a single pass, it is advantageous to run the code over several passes so that computationally intensive steps can be saved and adjustments made prior to subsequent processing. The following workflow was used to perform the wavelet-based analysis in Sturtevant et al. 2016 (included in the software package).

1. Gap-fill the data if planning to wavelet transform.
2. Create surrogates without any other processing (Config\_PreProcess\_createSurrogates\_ProcessNetwork.m)
   1. I recommend to create random walk surrogates (opts.SurrogateMethod = 4) if planning to wavelet transform the data (and the surrogates). They create random variation at all time scales that when wavelet decomposed along with the data will create random variability at the same timescale as the data. If not planning to wavelet decompose the data, I recommend to create random walk surrogates with wavelet variance matched to that of the data (opts.SurrogateMethod = 5). This method takes the longest to produce, but provides the best random data to accurately characterize the bias in mutual information when the sample size is low relative to the timescale of variability (almost always the case for environmental data).
   2. Save the output (opts.savePreProcessed = 1)
   3. I’ve found 50 surrogates to be sufficient (opts.nTests = 50)
   4. It’s best to create the surrogates without any other processing so that the wavelet transform used in later steps acts on the same set of ‘raw’ data. If surrogates and wavelet transform are done together, then the wavelet transforms at different scales are performed on a different set of surrogates. This is probably not that big a deal, but it means there is a slightly different process for surrogate data as opposed to actual data (the actual data remains the same for each wavelet scale while there is a whole new set of surrogate data for each wavelet scale).
3. Wavelet decompose all data, including surrogates (Config\_PreProcess\_waveletTransform\_ProcessNetwork.m)
   1. Use the surrogates in the files from here on out (opts.SurrogateMethod = 1)
   2. Choose the wavelet scale(s) (e.g. opts.waveN = 4 to extract variations on the order of 2^4 time steps). A vector can be input (e.g. opts.waveN = 4:6 to extract variations from 2^4 time steps to 2^6 time steps). The wavelet scales used in Sturtevant et al. 2016 were (half-hour time steps):
      1. Hourly: opts.waveN = 1:2
      2. Diel: opts.waveN = 3:6
      3. Multi-day: opts.waveN = 7:10
      4. Seasonal: opts.waveN = 11:14
   3. Save the output (opts.savePreProcessed = 1)
   4. Rerun the code for as many scales or sets of scales you want to extract. For example, to pull out hourly variation with half-hour time steps, first run and save output using opts.waveN=1:2. Then to extract diel variation, run and save the output again using opts.waveN=3:6.
   5. The reason why I don’t bin as well here is because binning is very computationally inexpensive, whereas the wavelet decomposition is very expensive. Saving the output at the wavelet decomp stage enables you to play with more options without having to rerun the computationally expensive wavelet decomp step.
4. Remove original gaps from *both* Data and Surrogates, and truncate the dataset to whatever you want to analyze (and save the output for further processing)
   1. This is done outside the ProcessNetwork code.
   2. For seasonal wavelet scale and half-hourly time steps, recommend to remove ~2-3 months at the very beginning and ends of the original wavelet decomposed dataset to avoid edge effects. For multiday, remove ~2-3 days on either side. For daily, remove ~3 hours. For hourly, remove 1 hour. The la8 wavelet transform has minimal edge effects and you could probably get away with removing less, but I like to be conservative if possible.
5. Run binning and entropy calculations (Config\_EntropyOpts\_ProcessNetwork.m)
   1. Use locally bounded bins (opts.binType = 1)
   2. Never trim the data (always set opts.trimTheData = 0). This destroys time lags.
   3. Use Surrogates in loaded files (opts.SurrogateMethod = 1)
   4. If there are a large number of gaps in the data, surrogate test each lag (opts.SurrogateTestEachLag = 1). This will help account for bias in the results due to sample size.
   5. Choose lag spacing (in increments of half-hours) appropriate to the scale. Recommendations:
      1. Hourly = ½ hour
      2. Daily = ½ hour
      3. Multiday = ~4 hours
      4. Seasonal = ~1-3 days
   6. If surrogate testing each lag the processing time is significantly longer. If it gets annoying and you don’t care about the mutual information between explanatory variables (e.g. soil temp with air temp), there is an option to choose ‘target’ variables (opts.targetVars). The entropy calcs will only be computed for these variables, using all other variables as explanatory variables. Set this option by inputting the names of target variables in opts.targetVars (e.g. opts.targetVars = {'FCH4\_gf', 'LE\_gf', 'NEE\_gf', 'gpp\_ANNnight', 'reco\_ANNnight'}). These will be a subset of opts.varNames. I added this option to the input arguments for Config\_EntropyOpts\_ProcessNetwork.m. If you don’t want to use it, simple input an empty cell (targetVars={}).
   7. If you have the parallel computing toolbox, I highly recommend using parallelization for entropy calcs (opts.parallelWorkers >= 2). It parallelizes over each lag and will cut computation time significantly. If you’re unsure, set opts.parallelWorkers = 2. If you get an error, move back to 0 or 1. If you don’t get an error, see how high you can go!
   8. Recommend using 99% significance threshold if testing lots of lags
   9. Save entropy calcs (opts.saveProcessNetwork = 1), and optionally the ‘preprocessed’ data (opts.savePreProcessed = 1) if you want to look at binned data
6. Plot output
   1. Use the plotting options at the end of one of the sample config\_runscripts.
   2. If there are lots of lags in the data, some new statistics may resolve harmonics in the lag plots that result from variations in sample size (this is still an active area of development – not done in Sturtevant et al. 2016). In this case, plot the relative mutual information normalized by the mean random IR (this is why we surrogate tested each lag). Note that for these new statistics to be useful, the surrogates must match the same sample size and timescale(s) of variability as present in the actual data (see comments about surrogate generation above).
      1. The mean IR (relative mutual information) of the surrogates at each lag *subtracted from* the IR of the actual data at each lag. This statistic has shown the most promise in removing the bias in mutual information when the sample size of the data is low relative to the timescale of the variability (e.g. seasonal variation). In other words, it removes the mutual information that is present even between completely random timeseries that have the same sample size, placement of gaps, and timescale of variability as the actual data. For the coupling lag plot (couplingLagPlot.m) and multi-coupling synchrony plot (multiCouplingSynchronyPlot.m), the new options are:
         1. popts.testStatistic = 'IRsubtractShuff’
         2. popts.SigThresh = 'SigThreshIRsubtractShuff'
      2. Relative mutual information *divided by* the mean IR of the surrogates (at each lag). This statistic is more a measure of how statistically significant the computed IR is (relative to completely random timeseries). For the coupling lag plot (couplingLagPlot.m) and multi-coupling synchrony plot (multiCouplingSynchronyPlot.m), the new options are:
         1. popts.testStatistic = 'IRnormByShuff’
         2. popts.SigThresh = 'SigThreshIRnormByShuff'