This code implements Granger causality conditioned to a limited subset of n_d variables, chosen each time as the most informative for each candidate driver.
Mutual information is computed with the covariance matrix in the Gaussian approximation.
The code can be modified according to your favorite recipe for MI calculation.

The first function to be run is init_partial_conditioning_par

input:

1. *data*, with dimensions (npoints nvar) or (npoints ntrials nvar) in case you have data divided in trials. The code adapts to your choice.
2. *ndmax* , the maximum number of variables to consider as simultaneously contributing. As a rule of thumb you can choose *ndmax* = nvar/2 for nvar<100 and reduce this fraction to nvar/20 for 1000 regions, but this of course depends on the nature of the system.
3. *order*, as the order of the autoregressive model, to determine with your favorite criterion (AIC, BIC, crossvalidation etc)

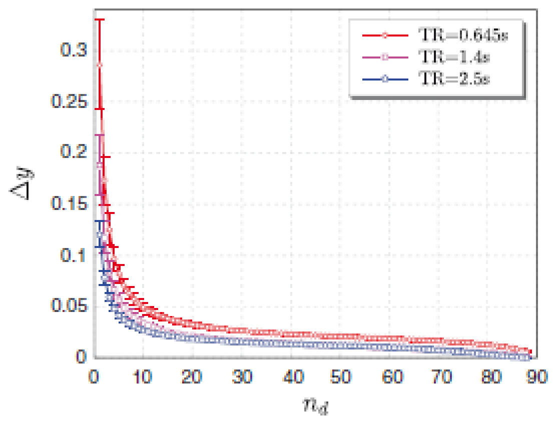
output:

1. y, the information
2. ind, for each candidate driver, the *ndmax* most informative variables, ordered.

When the program has finished running, you plot the incremental information when you add an additional variable n_d*+1* is included in the conditioning set.

```
plot(1:ndmax-1,diff(y'));
```

This will look like that



You can simply look for the knee of the curves (averaged among regions), or devise a fancier way to stop.

This step is quite time consuming, you better start with a low value of *ndmax*, then increase it if the knee is not reached.

Then you run partial_CGC_fix_nd_new with input:

1. *data*, as before
2. *order*, as before
3. *nd*, as the value selected as the knee of the curve

4. *ind*, the output of the previous function
output:
1. *pcgc*, the partially conditioned Granger causality matrix, where *pcgc(i,j)* indicates GC influence from *i* to *j*

The code exists both in matlab version (NAME_m.m) and in C to be compiled (NAME.cpp). They will need to be compiled using the command (all on the same line)

```
mex -largeArrayDims NAME.cpp
'matlabroot\extern\lib\win64\microsoft\libmwblas.lib'
'matlabroot\extern\lib\win64\microsoft\libmwlapack.lib'
```

Where *matlabroot* is the output of matlabroot command, and then you have to search where the two libraries above are.

The compiled version is in general faster, only for a limited number of variables (both total and n_d) the matlab version with parallel for can be slightly faster.

In its matlab version the code uses the parallel for loop parfor. In case of older matlab versions, parfor can be changed to standard for.

If you use the code please drop me an email (daniele.marinazzo@gmail.com) so that I can keep you posted with updates, bug fixes, etc