Skoltech
Skolkovo Institute of Science and Technology

The *Dos* and *Don’ts* of Using Causal Inference Analysis Through Time Series in Geosciences Applications

Jean Carlos Andrade De Carli

Moscow

July 2021

Copyright

Acknowledgments

I would like to acknowledge the effort and support of my friend and colleague Dmitry Voloskov. This work could not be proceed without his joint research and enthusiasm. For his time, attention and help I am deeply thankful.

I am sincerely thankful to Prof. Dimitri Pissarenko, who introduced me to the research topic of causal inference. Who, even though is superior in ranks, knowledge, and wisdom, treated me as equal and gave me all the freedom to persevere with my ideas and dreams. Prof. Pissarenko was always clear, frank and honest in our conversations and in his advice. And for these, I will be forever thankful for his chivalry and friendship.

Preface

Since the beginning of time, we have been motivated by “why” questions. These questions came along with observations, doubts, and willingness to intervene in a system to observe the outcome. This is a part of human nature. And perhaps this interesting aspect of our brains had made us the dominant species on this planet. Since we are born, we are intrigued about how things work. Why the sky is blue? What causes this effect? The willingness to obtain answers about natural phenomena is embedded into our intellect, and it is the main driving force of the academic society.

Science has been able to aid our desire to obtain answers about our planet for a long time. The development of physics and calculus in combination with modern computers encourages us to seek further challenges. One of them is the study of causes and effects. Better known for Causality, Causal Discovery, Causal Inference and Causal Reasoning.

This report is dedicated to inform and guide newcomers’ geoscientists to the research field of Causal Inference. I decided to build this guide specifically to support you to start quickly and advance fast into the main algorithms in Python script used for Causal Inference via time series. This report is rather a roadmap for *what to do, how to do, what to read, and what do to next* than a theoretical and philosophical approach about Causality. I believe that if you are reading this report, you would already have a clear picture of the power and why to study causality. Therefore this report was built to help students of Causal Inference to get a hands-on experience with advanced technics and codes in how to infer causal links through time series.

Table of contents

List of Figures

Technical Abbreviations

IDE	Integrated Development Environment
Tg	Tigramite* library itself or processes related to it.
HPC	High Performance Computing
ML	Machine Learning
AI	Artificial intelligence
np	Numpy* - python library
pd	Pandas* - python library
Parcorr	Partial Correlation test
MRC	Main Research Code*
DAG	Directed Acyclic Graph
TSG	Time Series Graph
PCMCI	Peter Clark Momentary Conditional Independence algorithm

*For further explanation, see section Glossary of Definitions

Glossary of Definitions

Geo-parameter: A geological parameter which is only control by the nature and the genesis of the rock formation. Such as porosity, permeability, particle size, mineral content, etc.

Numpy: Python script fundamental package for scientific computing. For more information, visit `numpy.org`.

Pandas: open data analysis and manipulation tool developed in Python script. For more information, visit `pandas.pydata.org`.

MRC: Main Research Code, refers to the main code developed to be used with the implementation of time series analysis via Tigramite (described in section 4.4.)

Tigramite: Python library for causal inference in time series. For more information, visit its documentation (`https://jakobrunge.github.io/tigramite`) and its GitHub repository (`https://github.com/jakobrunge/tigramite`).

1. Introduction

Causality, also known as causation, and causal inference is a descriptive study of cause and effect. The study of how variables behave and act on others to generate another variable or their value. Dr. Judea Pearl, one of the most prominent and contributor of causal inference research, says that the first step to think about causal inference is to consider that we live in a non-linear world. And that even a very simple mathematical equation should be written as non-linear. Generally we would write $F=m.a$, therefore $m.a = F$. Dr. Pearl suggests a mental exercise where instead of this linearity we think and write $m.a := F$ ($m.a$ "causes" F), and F can not cause ($F:=m.a.$) and influence in values of m and a.

When talking about causality, one of the first things that come to mind is its application on future perspectives, in other words, predictions and probabilities of something causing something to happen. And almost intuitively, the first question that comes is:

Why not solve forecast challenges by using a classical statistical approach?

The difference between these two domains of statistical learning and causal learning is the amount of data used. Statistics models will be validated as they were given an infinite amount of data, thus making statistical learning not so difficult. But in causal inference this step does not depend on data size, this step from measured data to causal model will never be trivial given that there will be always hidden variables, systematic causal errors and interdependencies among variables over time.

Causal inference gathers a couple of different approaches towards causal studies. Each of them generally has been designed and are been used in some specific field accordingly to the data provided. In our case, we decided to use causal inference in time series due to the closest relationship to the data used for this process and the nature of the data available from simulations and exploration from petroleum fields.

Causal inference in time series is used to reconstruct the complex network of observable and hidden variables in our system. As illustrated in figure 1, the aim is to reconstruct the underlying causal dependencies. We start from 1A where we have our data provided in time series, and search and calculate the best causal model (1B) to unveil the causal links and their directions as well as to present spurious associations (gray-arrows) which emerge because of common drivers or transitive indirect paths among variables. The ultimate goal is to construct the Time Series Graph (1C), which will describe time dependency among the variables over time.

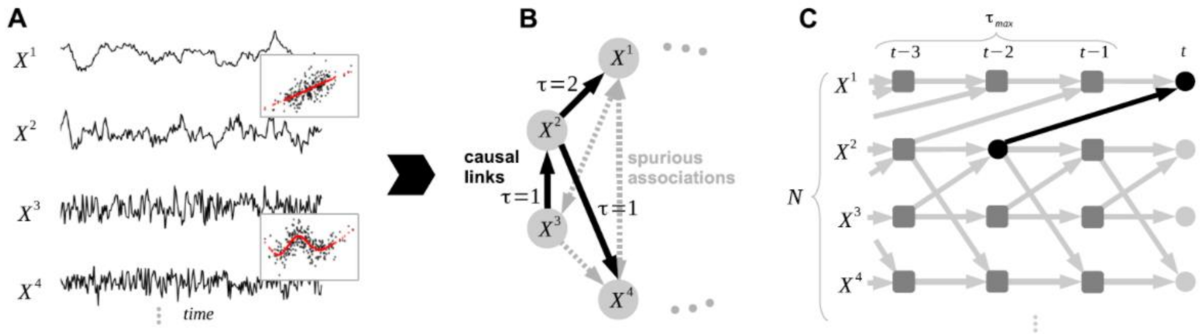


Figure 1: Causal inference network reconstruction from data series (Runge, 2018)

Causal inference has been a subject of study and development in several fields of research and industrial applications. However in geosciences, in particular to petroleum engineering, there are no publications (up to the creation of this report) presenting causal relationships between variables in the system. Therefore making this work extremely difficult and challenging. Bertoncello et al (2020), perhaps is the closest of causal inference to petroleum engineering application that has been published. However, their approach is focused on a more financial point of view (econometrics) rather than unveiling causal links and drivers into the system. The petroleum engineering industry, like other industrial fields, can benefit greatly from the integration of knowledge and scientific applications of different technologies and methods. Valuable reservoir models and field data come in time series format. To analyze these variables and events in a causal framework scheme, we used a state-of-the-art causal inference algorithm created by Runge et al. (2019). This time series analysis approach (named PCMCI) has been used successfully in many studies from different fields of expertise. PCMCI (Runge et al. (2019)) and its extension PCMCI+ (Runge (2020)) have been used in different fields of research. These studies have been applied mainly in Earth systems, particularly in investigations of global climate events (Krich et al. (2020), Krich et al. (2021), Runge et al. (2015)).

Dr. Jakob Gerhard Bernhard Runge has been working on the development of a causal inference approach via time series for over 10 years. Even though some other authors claim to have an equally powerful approach and algorithms for unveiling causal relationships among variables in the system. Dr. Runge, with no doubt, has the most well-constructed online community environment and the best visualization tool for the calculation results. During the past decade, Dr. Runge and his team have been dedicated to the development of a Python Library named Tigramite which helps the users to calculate and estimate causal links by creating the Directed Acyclic Graph and the Time Series Graph of the system.

2. Literature – what to read

As I have mentioned, this report is a roadmap for literature to be known and a hands-on guide into the processing part of causal inference in time series. For beginners, I strongly recommend this flowchart pathway.

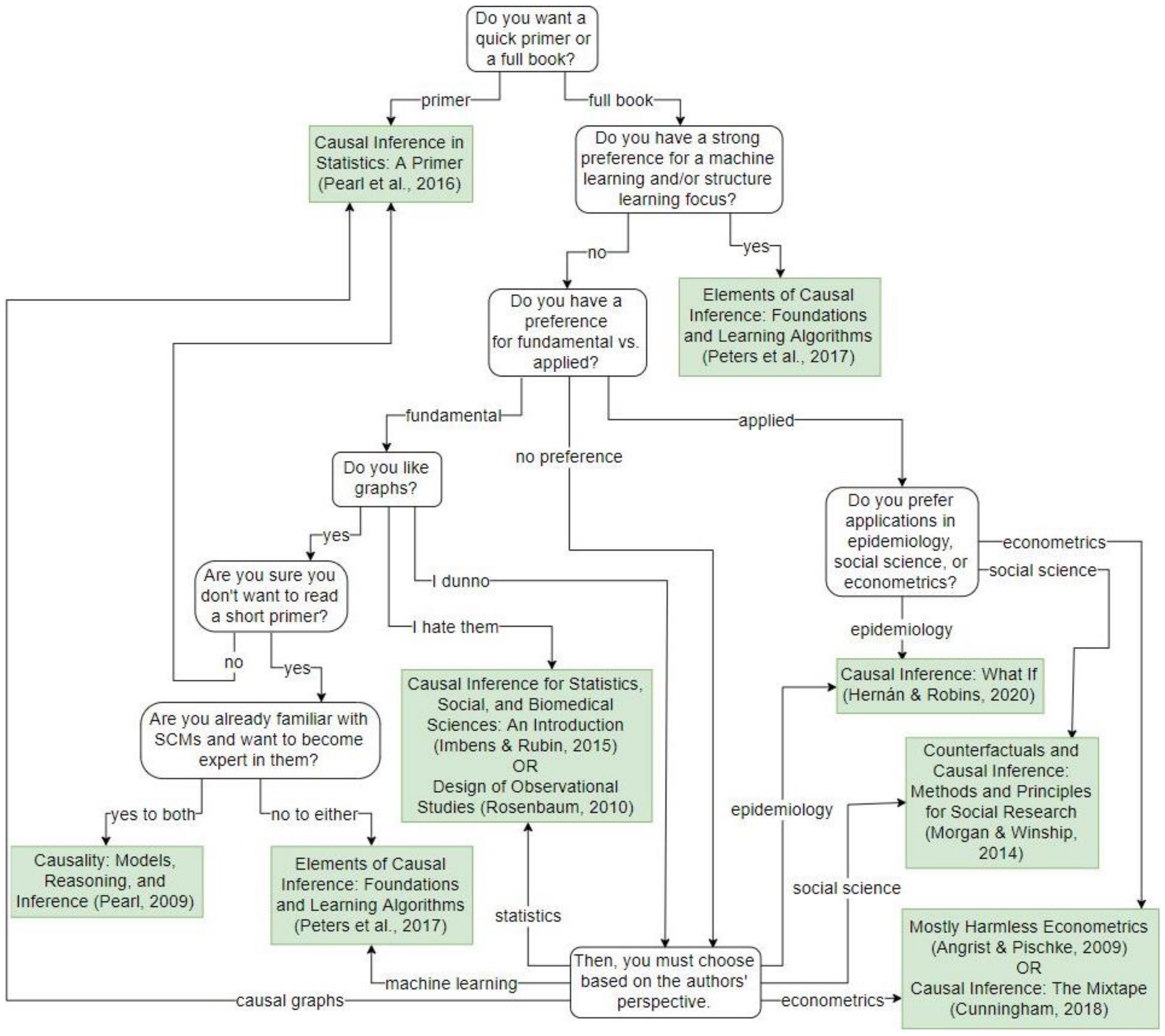

Do you want a quick primer or a full book?
primer
full book
Causal Inference in Statistics: A Primer (Pearl et al., 2016)
Do you have a strong preference for a machine learning and/or structure learning focus?
no
yes
Do you have a preference for fundamental vs. applied?
Elements of Causal Inference: Foundations and Learning Algorithms (Peters et al., 2017)
fundamental
applied
no preference
Do you like graphs?
Do you prefer applications in epidemiology, social science, or econometrics?
econometrics
social science
yes
I dunno
I hate them
Are you sure you don't want to read a short primer?
epidemiology
no
yes
Causal Inference: What If (Hernán & Robins, 2020)
Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction (Imbens & Rubin, 2015) OR Design of Observational Studies (Rosenbaum, 2010)
Are you already familiar with SCMs and want to become expert in them?
Counterfactuals and Causal Inference: Methods and Principles for Social Research (Morgan & Winship, 2014)
yes to both
no to either
epidemiology
Causality: Models, Reasoning, and Inference (Pearl, 2009)
Elements of Causal Inference: Foundations and Learning Algorithms (Peters et al., 2017)
social science
statistics
Mostly Harmless Econometrics (Angrist & Pischke, 2009) OR Causal Inference: The Mixtape (Cunningham, 2018)
Then, you must choose based on the authors' perspective.
machine learning
causal graphs
econometrics

Figure 2: How to start in Causal Inference (Neal, 2020).

Below, a list of literature which helped me to understand more about causal inference.

2.1. Causal Inference - books

- *The Book of why*. Pearl and Mackenzie (2018).
- *Causal inference in statistics: A primer*. Pearl, J., Glymour, M., & Jewell, N. P. (2016).
- *Causation, Prediction, and Search.* Spirts et al, (2000).
- *Elements of causal inference: foundations and learning algorithms.* Peters, J., et al (2017)

2.2. Causal Inference - articles

- *Using Causal Inference in Field Development Optimization: Application to Unconventional Plays*. Bertoncello et at (2020).
- *Conditional independence testing based on a nearest-neighbor estimator of conditional mutual information*. Runge (2018).
- *Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets.* Runge (2020).
- *Detecting and quantifying causal associations in large nonlinear time series datasets.* Runge et al (2019).
- *Causal Network Reconstruction from Time Series: From Theoretical Assumptions to Practical Estimation*. Runge et al (2018).
- *Identifying causal gateways and mediators in complex spatio-temporal systems*. Runge et al (2015).
- *Quantifying information transfer and mediation along causal pathways in complex systems*. Runge (2015).
- *Detecting and quantifying causality from time series of complex systems*. Runge (2014).

2.3. Supplement material

Books:

- Functional Programming in Python. Mertz (2015)
- Bayesian analysis with Python. Martin (2018)
- Learning Bayesian networks. Neapolitan (2004).
- Applied Bayesian Modelling. Congdon (2003).
- Bayesian statistics for Beginners, a step-by-step Approach. Donovan and Mickey (2019).
- Bayes' Rule. A tutorial introduction to Bayesian analysis. Stone (2013).
- Practical time series analysis. Nielsen (2019).

Online courses:

- Applied time series analysis in Python. Udemy. Peixeiro (2021).

3. Reservoir model – synthetic data

The 3D reservoir simulation employed is an expansion of the work of Voloskov and Pissarenko (2020), who applied their approach at a 2D grid level. This 3D two-phase immiscible flow hydrocarbon-bearing reservoir model is created and compiled accordingly to mass conservation equations for oil and water. Features of the reservoir such as compressibility, gravitational force, and capillary pressure are neglected (Voloskov and Pissarenko, 2020). The design of this reservoir simulation grants the power for the user to easily set the location and geometry of the wells (injectors and producers), in addition to grid model properties (grid and cell sizes, porosity, permeability, original water, and oil, in place). Wells can be set totally vertical, or partially vertical with a horizontal segment. Users can change the diameter, direction (azimuth), and length of the wells. This reservoir simulator is a great tool to investigate the performance of the horizontal perforated segment of the well into the hydrocarbon-bearing grid layer. With this model, it is possible to observe how the diameter and horizontal length of the production well can change the values of fluid (oil and water) production. Thus, generating time series of injection regime and production values of oil and water until the given maximum time step.

The main source code for this reservoir model is under development in Python and it is still under test by Dmitry Voloskov. Any questions regarding new versions, updates and features of the simulations should be addressed to the author.

Synthetic 3D reservoir model (figure 3) and porosity distribution for each layer. The reservoir model is composed of 3 layers, 4 injection wells, 1 production well with a horizontal segment located at the second layer (layer 1), and a low porosity trend region from the top-left corner towards the lower-right corner of the grids.

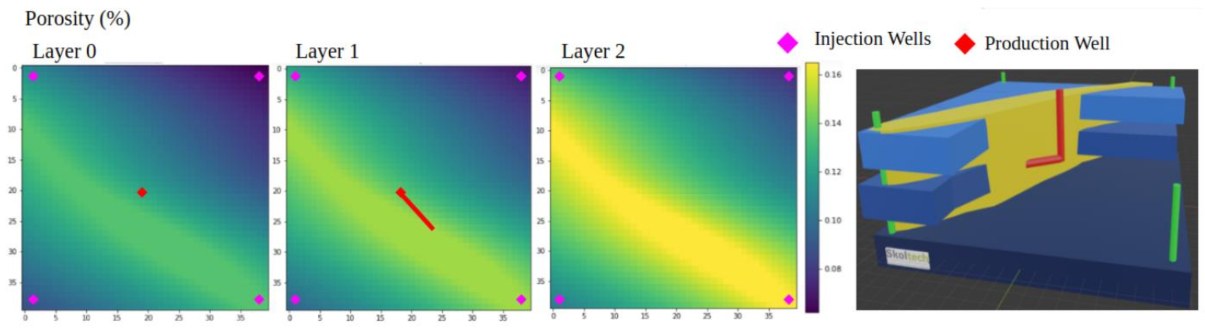


Figure 3: Synthetic 3D reservoir model, porosity distribution for each layer and well locations.

Figure 4, shows images of a simulation from the early stages of water injection (time step 100) up to the end of the injection regime (time step 2000). Note that each injection well (from each corner) presents a different shape in its flooding front. This effect is created due to the different injection rates from each well, in addition to the different porosity distribution in the system.

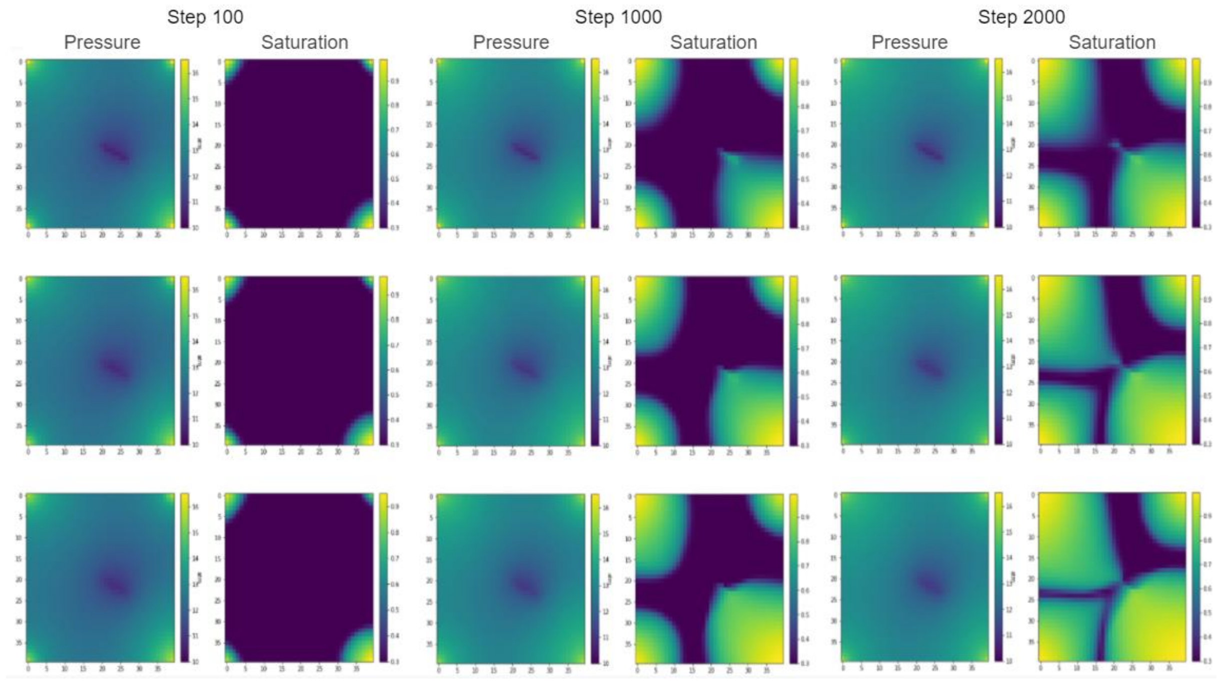


Figure 4: flooding front from each injection well from the early stages of injection (step 100) until the end (step 2000).

4. Data Processing and Python codes

4.1. Data imported

We use the reservoir simulator (described in section 3) to generate 10 different scenarios to import to the analysis of causal inference with times series (figure 5). Each scenario had either different injection rates or different BHP from each other. With exception of the simulation *data Poro Homo* that has homogeneous porosity. Each particular aspect of every single simulation can be found through the Main Research Code (MRC, described in section 4.2.). The reservoir 3D grid has a size of 40x40x3 (figure 3). It is composed of 3 layers, named accordingly to their indices (layer 0, layer 1, and layer 2). All of them bearing different values of original water and oil in place. Production wells in all cases were perforated in the second layer of the grid. The physical characteristic (location, orientation and diameter) of injection and production wells where constant in all scenarios. Injection wells are placed in the first grid cell corner, and production wells placed in the center of the grid. Injection wells were labeled inj.0 (top-left), inj.1 (lower-left), inj.2 (top-right) and inj.3 (lower-right).

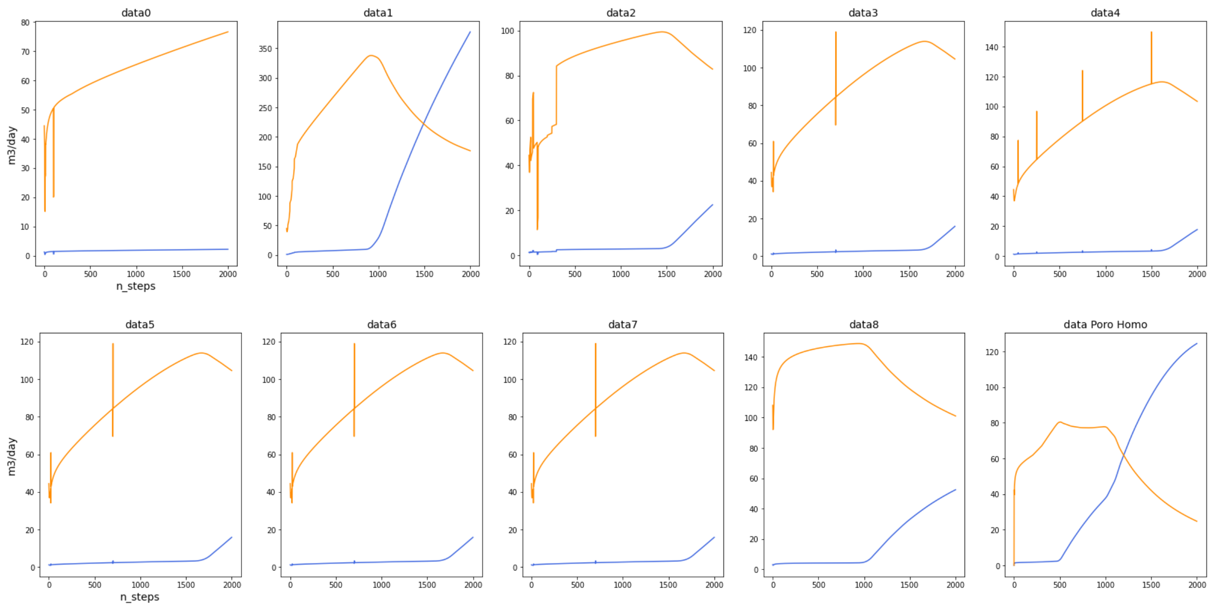


Figure 5: 10 different reservoir simulations used in this report and in the MRC.

4.2. Tigramite data processing and the Main Research Code (MRC)

Tigramite is a library develop for python script that is used for causal inference analysis through time series data. The library efficiently reconstructs the Directed Acyclic Graph (DAG) and the Time Series Graph (TSG) of a given dataset. It is based on linear as well as non-parametric conditional independence tests applicable to discrete or continuously-valued time series. With the obtained results, Tigramite allows us to model causal dependencies for causal mediation and prediction analyses.

In order to illustrate from start to end of the data processing in this time series causal discovery approach. I present, below, my personal workflow from the moment you have your data available until you obtain the results (Runge, 2019).

Tigramite provides several causal discovery methods that can be used under different sets of assumptions. An application always consists of a method and a chosen conditional independence test (Runge, 2021 – `Tigramite repository`). The Conditional Independence tests implemented are: Parcorr, GPDC, CMIknn and CMIsymb.

Figure 6 shows the overview of how Tigramite works from the beginning to the end. The figure shows the steps of calculations and what the user should do in each step.

Processes:

I. Convert your input data to Numpy array format. Data often comes in different formats (.txt | .csv | .dat), when importing to the python environment to be used with Tigramite. The data must be converted to numpy arrays. Tigramite will create then a dataframe format accordingly to its requirements for data processing.

II. Choose a Conditional Independence Test. Tigramite will run its PCMCI or PCMCI+ causal inference algorithm with the Conditional Independence test selected (Parcorr, GPDC, CMIknn or CMIsymb).

III. Results. Tigramite will generate two main outputs, the TSG and the DAG. You can change easily the plotting features to better visualize the structure and values of the causal links.

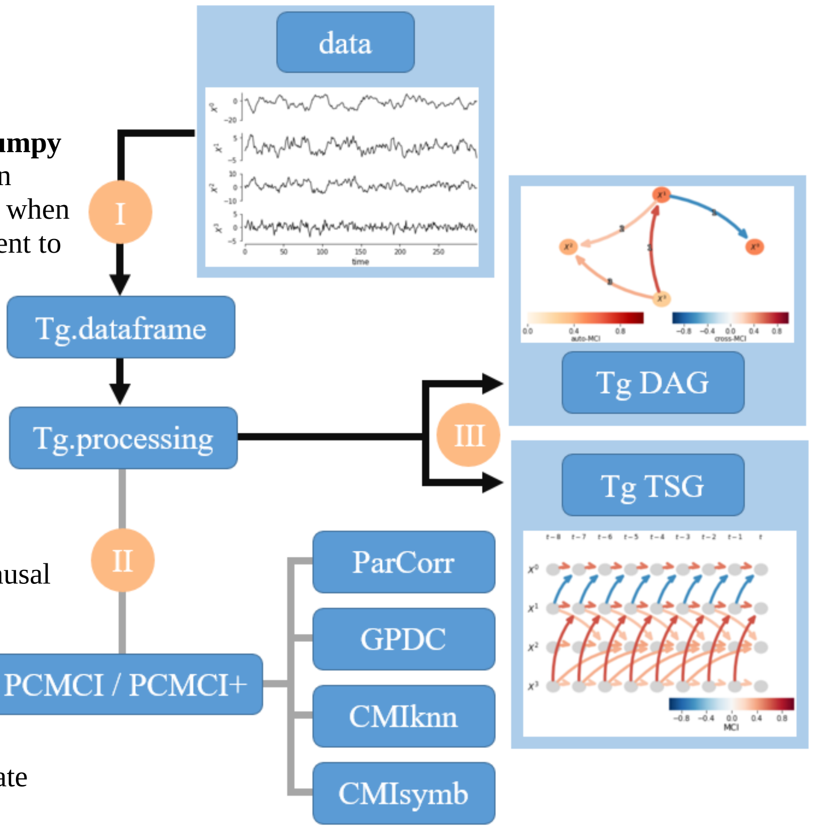


Figure 6: Tigramite data process flowchart.

4.3. Conditional Independence Tests

4.3.1. ParCorr

The **Partial Correlation** test (Parcorr) is a conditional independence test based on the Linear Ordinary Least Squares regression and it is a conditional independence test for non-linear Pearson correlation on the residuals. The original source for this implementation in Tigramite can be found `here`. An example of ParCorr with data generated by `np.random.seed` (figure 7) can be find at the MRC under the defined function `jtg_ParCorr_eg` (annex 01). And its implementation with our synthetic data context at Annex 02 and in the MRC under de defined function `jtg_ParCorr`.

ParCorr assumptions: univariate, continuous, linear, Gaussian dependencies.

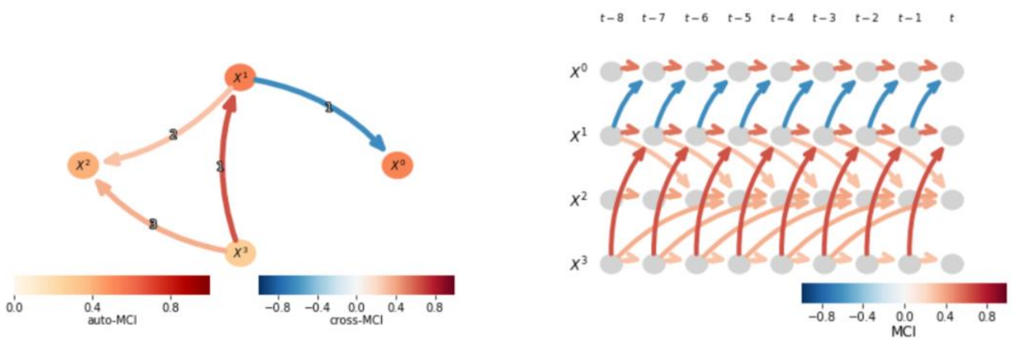


Figure 7: DAG and TSG of *np.random.seed* data processed with PCMCI and ParCorr.

4.3.2. GPDC

Conditional independence test based on Gaussian Processes (GP) and Distance Correlation (DC) on the residuals (Székely et al 2007). GP is estimated with `scikit-learn` and it allows to flexibly specify kernels and hyperparameters or let them to be optimized automatically. The distance correlation (DC) is implemented with `cython`. An example of GPDC implementation with data generated by `np.random.seed` (figure 8) can be find at the MRC under the defined function `jtg_GPDC_eg` (annex 03). And its implementation with our synthetic data context at Annex 04 and in the MRC under de defined function `jtg_GPDC`.

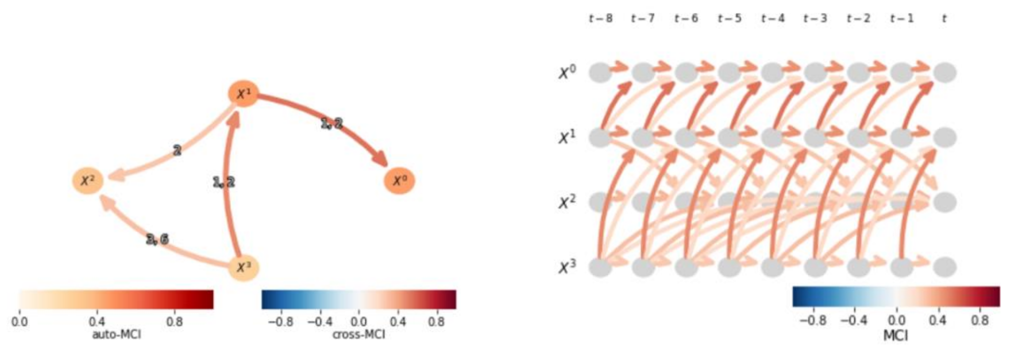


Figure 8: DAG and TSG of *np.random.seed* data processed with PCMCI and GPDC.

GPDC assumptions: univariate, continuous, additive dependencies.

4.3.3. CMIknn

The most general conditional independence test implemented in Tigramite is CMIknn based on conditional mutual information estimated with a k-nearest neighbor estimator. This test is described in the paper (Runge, 2018). Conditional mutual information is the most general dependency measure coming from an information-theoretic framework. It makes no assumptions about the parametric form of the dependencies by directly estimating the underlying joint density. The test here is based on the estimator in

S. Frenzel and B. Pompe, Phys. Rev. Lett. 99, 204101 (2007), combined with a shuffle test to generate the distribution under the null hypothesis of independence. The knn-estimator is suitable only for variables taking a continuous range of values. For discrete variables use the CMIsymb class (Runge 2021, Tigramite documentation).

Knn nearest neighbor regression bullshit.

$$I(X;Y|Z) = \int p(z) \iint p(x,y|z) \log \frac{p(x,y|z)}{p(x|z)\cdot p(y|z)} dxdydz$$

and the knn-estimator is given by

$$\hat{I}(X;Y|Z) = \psi(k) + \frac{1}{T}\sum_{t=1}^{T}\left[\psi\left(k_{Z,t}\right) - \psi\left(k_{XZ,t}\right) - \psi\left(k_{YZ,t}\right)\right]$$

being ψ a Digamma function, which it is a logarithmic derivative of the Gamma function given as:

$$\psi(x) = \frac{d}{dx} ln\, \Gamma(\mathrm{x})$$

An example of CMIknn implementation with data generated by `np.random.seed` (figure 9) can be find at the MRC under the defined function `jtg_CMIknn_eg` (annex 05). And its implementation with our synthetic data context at Annex 06 and in the MRC under de defined function `jtg_CMIknn.`

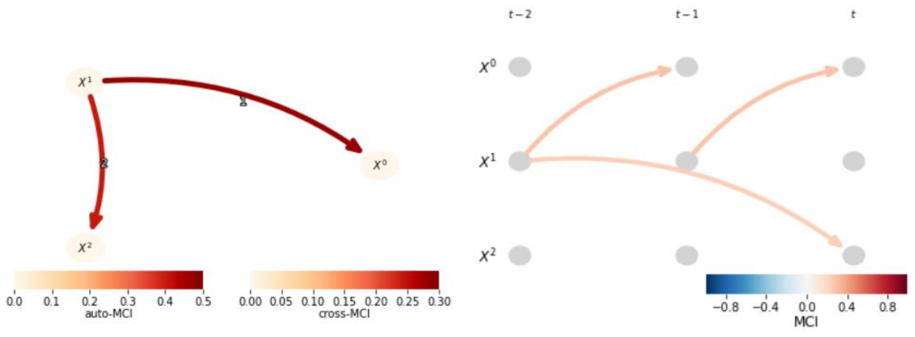


Figure 9: DAG and TSG of *np.random.seed* data processed with PCMCI and CMIknn.

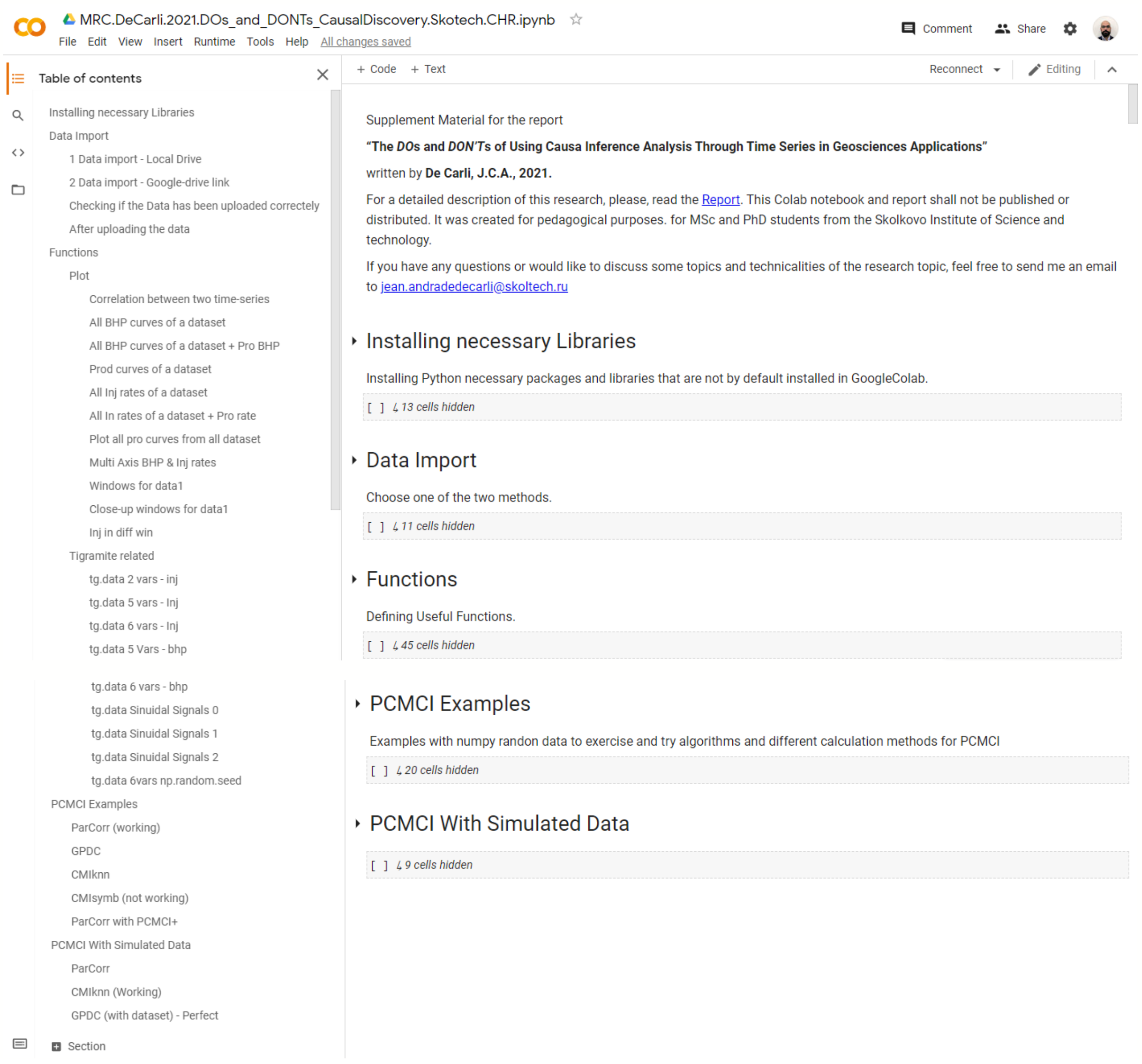
CMIknn assumptions: multivariate, continuous, general dependencies.

4.4. The MRC and to how navigate and use it

The **Main Research Code (MRC)** is a general and powerful tool for newcomers to causal inference, especially for those who are seeking a better understanding of how to use the Tigramite library. And find out for themselves about the causal relationship among their variables in a time series manner. The MRC was developed in python in the Google Colaboratory (Colab) environment (figure 10). For more information about Colab, *`visit`*. Get started with Colab `here`. Learn about *why* Colab was used for this project, see section **5.2. DOs.**

Nomenclature in the MRC

`j.` *"Why every function starts with 'j'?"*
This is a simple mechanism that allows the user to first type **j** and the autofill of the Python IDE will show all the functions defined in the notebook.
`jp.` Meaning that it is a plot function and will not store any calculation afterwards.
`Inj`$_n$ **`or I`**$_n$**`.`** Meaning or referring to an Injector well followed by its ID number. Example, `inj1`, injector 1.
`tg.` Meaning or referring to the Tigramite Library.
`Po`. Production of Oil.
`Pw`. Production of water.
`CP`. Cumulative production.
`vars.` Variables (nodes in the causal graphs).
`_eg.` Defined function which will generated a working example of Tigramite with a specific conditional independence test.


MRC.DeCarli.2021.DOs_and_DONTs_CausalDiscovery.Skotech.CHR.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect Editing
Table of contents

Supplement Material for the report

"The *DOs* and *DON'Ts* of Using Causa Inference Analysis Through Time Series in Geosciences Applications"

written by **De Carli, J.C.A., 2021.**

For a detailed description of this research, please, read the Report. This Colab notebook and report shall not be published or distributed. It was created for pedagogical purposes. for MSc and PhD students from the Skolkovo Institute of Science and technology.

If you have any questions or would like to discuss some topics and technicalities of the research topic, feel free to send me an email to jean.andradedecarli@skoltech.ru

Installing necessary Libraries

Installing Python necessary packages and libraries that are not by default installed in GoogleColab.

[ ] 13 cells hidden

Data Import

Choose one of the two methods.

[ ] 11 cells hidden

Functions

Defining Useful Functions.

[ ] 45 cells hidden

PCMCI Examples

Examples with numpy randon data to exercise and try algorithms and different calculation methods for PCMCI

[ ] 20 cells hidden

PCMCI With Simulated Data

[ ] 9 cells hidden

Figure 10: Screenshot of the MRC with notebook's chapters
and all the functions implemented (left-side colum, table of contents).

View and download the MRC `here`.

In the MRC you will find information about:

- How to update necessary Python libraries;
- How to upload the simulation data to the notebook, with two different methods.

And functions for:

j_corrcoef_2ts_plot	Correlation plot given two time series and a given lag	(figure 11)
jp_all_bhp_cruves	Plot a single image with all inj BHP from a dataset.	(figure 12)
jp_all_bhp_cruves_Pro	Plot a single image with all inj BHP and BHP of the Production well from a dataset.	(figure 13)
jp_prod_curves	Plot production curves (water rate and oil rate)	(figure 14)
jp_all_inj_curves	Plot a single image with all inj rates.	(figure 15)
jp_all_inj_curves_Pro	Plot a single image with all inj rates and production of water and oil.	(figure 16)
jp_all_prod_curves_datasets	Plot all production curves (oil and water) from all datasets imported.	(figure 5)
jp_multi_axis	Plot a single image with BHP of all inj wells and production curves (oil and water)	(figure 17)
jp_inj_pro_diff_win	Plot 3 images. 1st showing 4 inj rates together and separately. 2nd Production curves and a chosen inj well. 3rd Production curves.	(figure 18)

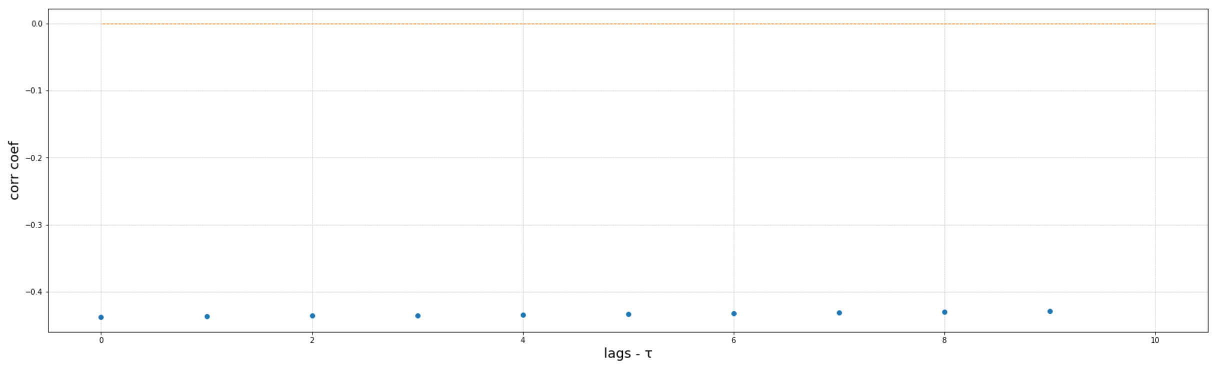


Figure 11: Correlation plot given two time series at a specific lag range.

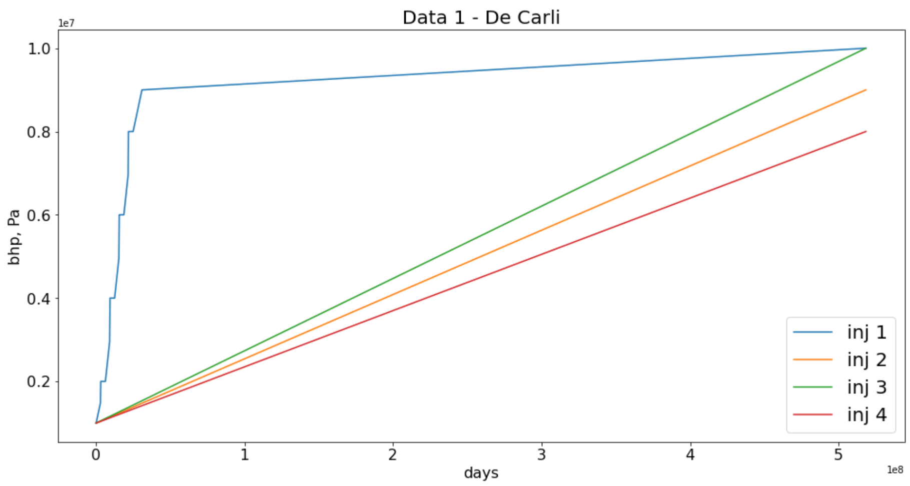


Figure 12: Plot with all injection wells BHP from a given dataset.

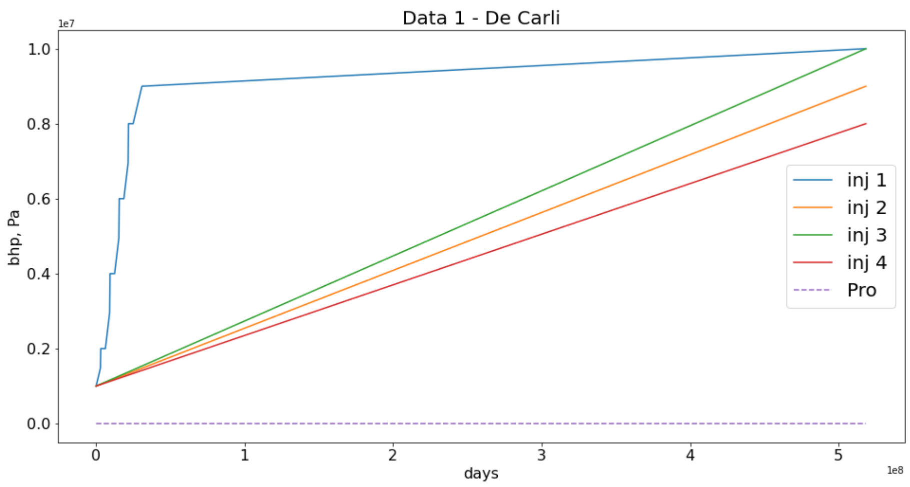


Figure 13: Plot of all inj BHP and Production well BHP.

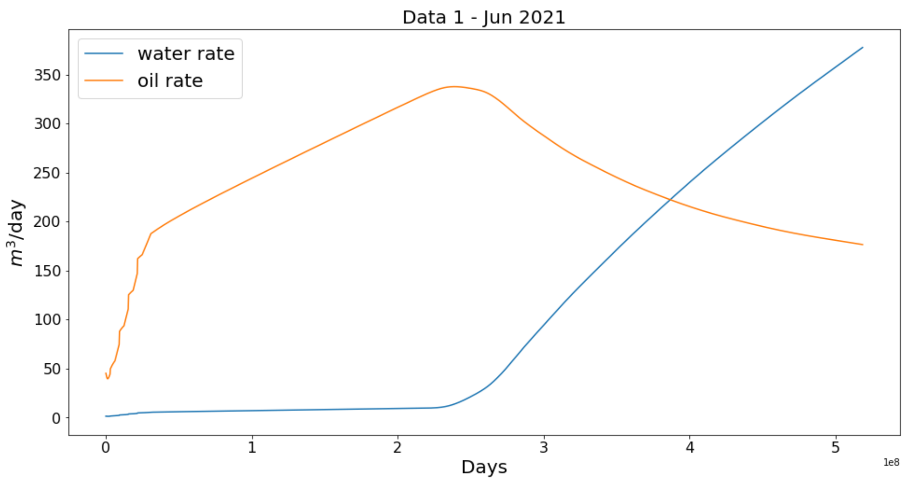


Figure 14: Production curves from a given dataset.

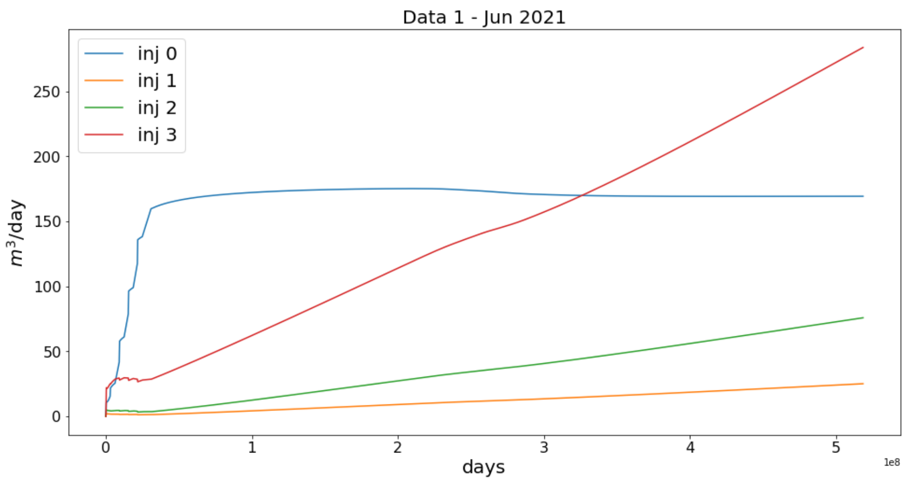


Figure 15: all inj wells rates from a given dataset.

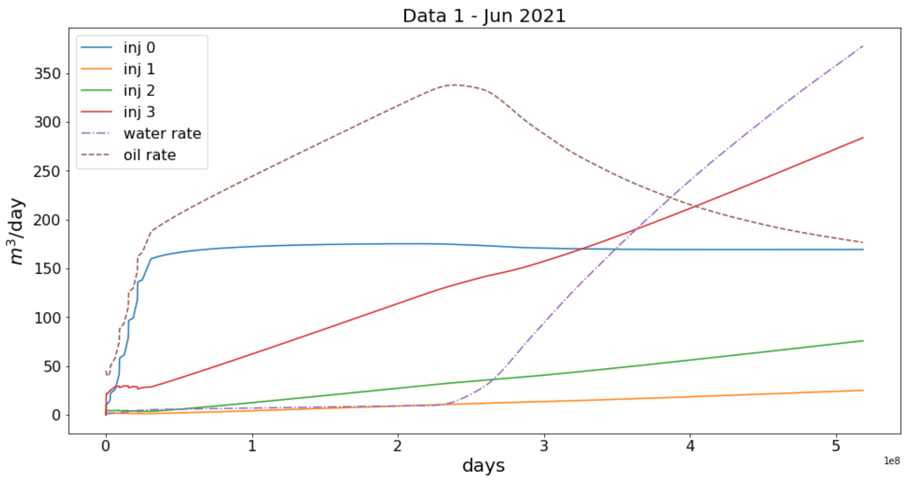


Figure 16: Injection and production curves from a given dataset.

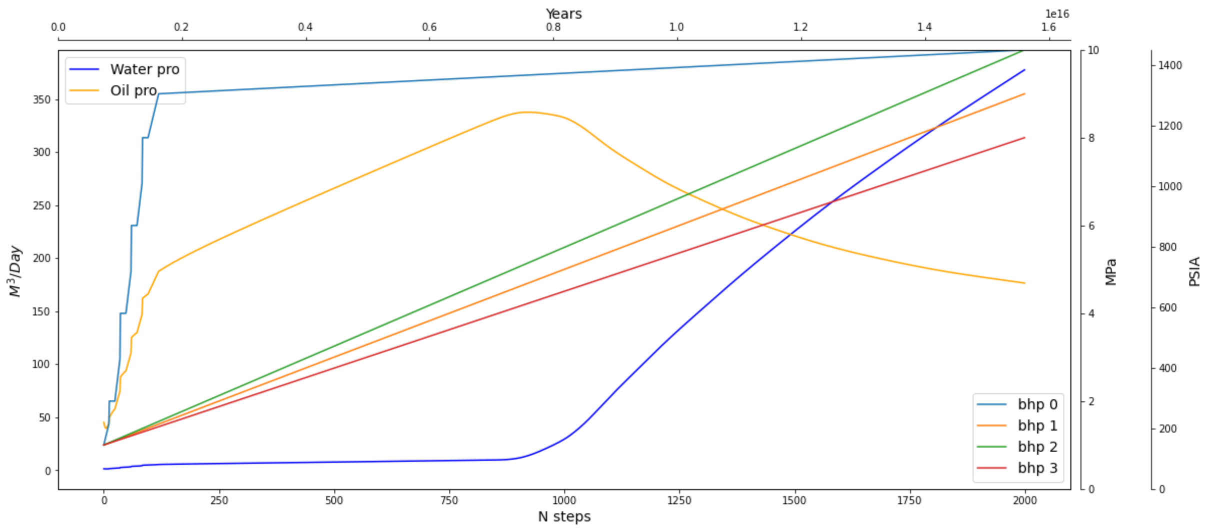


Figure 17: Production curves, injection curves and BHP.

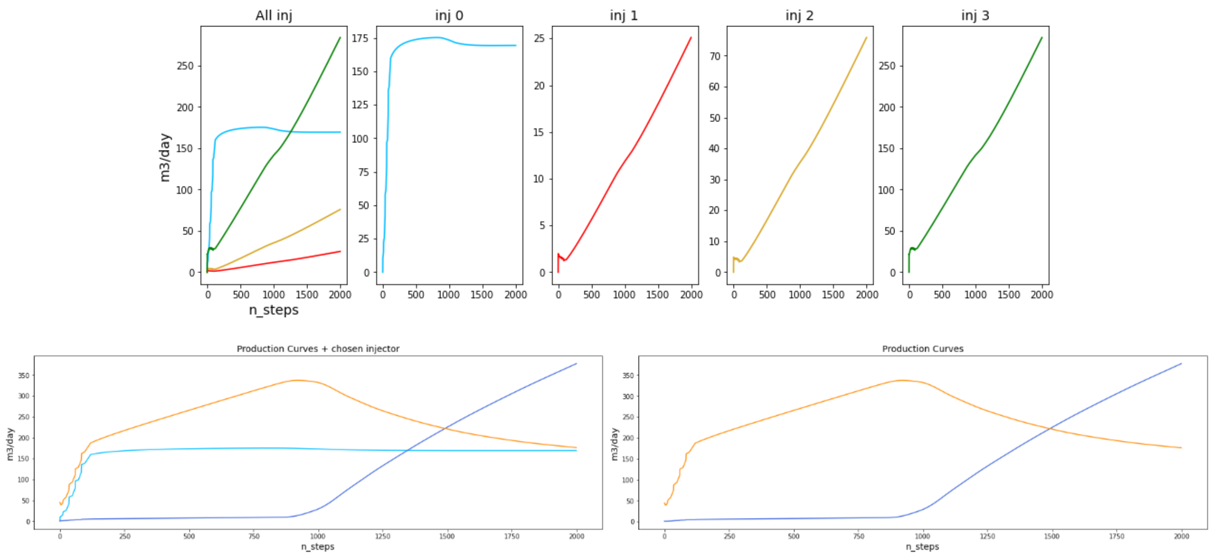


Figure 18: Plot 3 images. 1st showing 4 inj rates together and separately.

2nd Production curves and a chosen inj well. 3rd Production curves.

5. DOs & DONTs

5.1. DOs

DO. Conceptual

Understand, read and reflect.

- **Simpson's Paradox.**
- **Conditional Dependence tests.**
- **Reichenbach's Common Cause principle** (confounders).
- **Structured Causal Models.**
- **Causal Assumptions:**
    - **Faithfulness / Stableness;**

      Independencies in data arise not from coincidence, but rather from causal structure *or, expressed differently.* If two variables are independent given some other subset of variables, then they are not connected by a causal link in the graph.

    - **Causal Sufficiency;**

      Measured variables include all of the common causes.

    - **Causal Markov Condition;**

      All the relevant probabilistic information that can be obtained from the system is contained in its direct causes or, expressed differently, If two variables are not connected in the causal graph given some set of conditions (Runge, 2018 for further info), then they are conditionally independent.

    - **No Contemporaneous effects;**

      There are no causal effects at lag zero.

    - **Stationarity;**
    - **Parametric assumptions of independence tests.**

- **Causal effects mediation.**
- **Causal Inference Algorithmic approaches**
    - **Constraint-based:** Calculate independencies in the data and do backwards inference.
      It is used to minimize the degree of false negative edges.
    - **Score-based or Bayesian:** Calculate the likelihood of different DAGs given the data.
      It is used minimize the degree of false positive edges.
    - **Hybrids:** Combination of both (**a**, and **b**) at a certain degree.

- **The PC algorithm**
- **Granger Causality**
- **Time series**
    - Stationarity (Dickey-Fuller test). Seasonality. Autocorrelation. Mean. Variance. Covariance.

DO. Practice

Hands-on coding!

Causal inference calculations are mainly being developed openly via Python and R scripts. You should pick one, or two, and go to practice and code. The algorithms for causal inference are huge. You must be familiarized and skilled at least in one of these two programming languages. There are several libraries that can help us. One of them is Tigramite, which I have been using for close to 1 year. I strongly recommend the use of this package for time series analyses. Below, I list some of the other open-source python libraries that might interest you.

The size and time of calculations of the state-of-the-art causal algorithms are immense. Hence, it is crucial that you have skills and that you are willing to expand your code via HPC and parallel implementation. For python programmers, it is decisive that you master the tools with parallel implementation of your codes. An exceptional library to study and use, it the `mpi4py`. I have personally used this library for projects of Full-Wave-Form-Inversion (for seismic studies) and the calculation time saved is enormous. You can find an example of Tigramite in parallel computing in annex 07.

Additional Causal libraries in Python you should know.

`CausalML`	A python package for uplift modeling and causal inference with ML.
`DoWhy`	A python library for causal inference that supports explicit modeling and testing of causal assumptions.
`Py-Causal`	Python Application Programming Interface (API) for causal modeling algorithms.
`PyMC3`	Python API for probabilistic programming. Bayesian modeling and probabilistic ML.
`CausalNex`	Python library for causal reasoning with Bayesian networks.
`PyRo`	Deep universal probabilistic programming with Python and Pytorch.

5.2. DONTs

DON'T. Conceptual

Pay attention and analyze.

- **DO NOT:**
  - **Ignore causal analyses** into your project and dataset. Variables can represent a give a totally different understanding given other's variables dependencies and conditions.
  - **Jump into a "panacea algorithm".** Open source causal algorithms are developed for very specific tasks which might differ from your goals.
  - **Rush into a causal inference method** without reading and testing carefully its **causal assumptions** to your dataset.
  - **Think that causal inference will solve everything**. As I have seen, the studies in causal inference with constraint-based approaches are used as qualitative approaches. Later, you might need to implement a more elaborated ML algorithm to support your findings.

DON'T. Practicalities

Act, avoid mistakes!

- **DO NOT:**

- **Ignore online open discussion platforms.** Causal inference is one relatively brand-new research and development topic. Researchers are very communicative and open to reply to questions from students on several open platforms. Github *issues* and *discussions* might be one of the best.
- **Ignore social media.** Nowadays social media can be highly distracting and can lead to procrastination. However, do not ignore it. Many leading scientists in causal inference are very active in social media. Dr. Pearl, is highly active in his twitter account, as well as Dr. Runge and many other team members and PhD students from Dr. Pearl and Dr. Runge.
- **Work with non-stationary time series**. At least not before a new version of Tigramite. A Causal inference approach for non-stationary time series has been extremely requested in Tigramite Github repository. Several replies either from Dr. Runge or some of his research team members have replied enthusiastically with phrases such "we are working on that" and "[…] Stay tuned"**. Tigramite community on GitHub** is very active, I strongly recommend to participate.
- *`!pip install tigramite`*. Instead, use `pip install git+https://github.com/jakobrunge/tigramite.git` if you are using Tigramite via Colab. It has been reported for several users that the installation of Tigramite might lack in very small details

that will later create some problems whether you will use some calculations that will require cython. I personally like to work with Spyder IDE, I installed Tigramite in Linux (Ubuntu) and in Windows (Win.10) in two different machines and I had always problems with this issue. My gateway was to use and develop my codes in Colab. Problem solved. Currently, just a little detail with Tigramite and the version of the matplotlib installed by default in Colab. You can see and solve this issue by reading and running the first code lines in the MRC.

5.3. Addressing our challenges

In th petroleum engineering industry the data generated or gathered from surveys can be divided in two main group domains (static and dynamic). Let us call the first group "static" meaning that this data and its values do not change over time. This group can be defined as geological parameters (geo-parameters) such as porosity, permeability, mineralogy, stratigraphy and other. The geo-parameters share one main feature. All of them are created and driven by natural forces. They were created during rock-genesis. And most importantly, they are basically independent of time. At least to the time scale that we consider in our studies (a couple of years or decades).

At this point, one might ask: "But by using hydraulic fracturing technology, the permeability of the rock formations are changed from their original state. Therefore, why keep addressing these geo-parameters as static?"

By using enhanced oil recovery (EOR) methods, we indeed interfere in the original state of the rock formations. However, this change of permeability does not represent a significant event into a time series. To illustrate this point, let us think that we were given a dataset from a simulation of a single event of hydraulic fracturing in a reservoir model. It means that the permeability of the system can change once in the entire cycle of the simulation. This event can not be integrated in a causal inference framework and be treated as time dependent. At least not into the proposed data processing plan with Tigramite.

The second group called "dynamic" represents the data that are obtained by constant measurement over time. The data, variables, which are made and controlled by human interventions. For example, injection and production rates. All of them discretized over a minimum and a maximum number of time steps, the cycle of the exploration of the field. This group are the variables that we implement into a causal inference framework to discover their causal relations.

The greatest challenge for us is to integrate in a causal inference manner the analyses of static variables and dynamical variables. Without this integration, any further investigation in causal inference in petroleum engineering would be merely qualitative. And the results would not benefit the industry significantly.

A final piece of advice on my part, not just for the causal inference enthusiasts, but for all of us. Take the Simpson's paradox analysis into your life. "All that glitters is not gold."

6. Bibliography

Alberdi, X.A.T., Gerhardus, A., Eyring, V., Denzler, J. and Runge, J. (2021) Mapped-PCMCI: an algorithm for causal discovery at the grid level. Tech. rep., Copernicus Meetings.

Bertoncello, A., Oppenheim, G., Cordier, P., Gourvénec, S., Mathieu, J. P., Chaput, E., & Kurth, T. (2020). Using Causal Inference in Field Development Optimization: Application to Unconventional Plays. *Mathematical Geosciences*, *52*(5), 619-635.

Frenzel, S., & Pompe, B. (2007). Partial mutual information for coupling analysis of multivariate time series. *Physical review letters*, *99*(20), 204101.

Krich, C., Migliavacca, M., Miralles, D.G., Kraemer, G., El-Madany, T.S., Reichstein, M., Runge, J. and Mahecha, M.D. (2021) Functional convergence of biosphere–atmosphere interactions in response to meteorological conditions. Biogeosciences, 18(7), 2379–2404.

Krich, C., Runge, J., Miralles, D.G., Migliavacca, M., Perez-Priego, O., El-Madany, T., Carrara, A. and Mahecha, M.D. (2020) Estimating causal networks in biosphere–atmosphere interaction with the PCMCI approach. Biogeosciences, 17(4), 1033–1061.

Neal, B. (2020). Introduction to Causal Inference from a Machine Learning Perspective. *Course Lecture Notes (draft)*.

Oberst, M., Thams, N., Peters, J., Stontag, D. (2021). Regularizing towards Causal invariance: Linear Models with Proxies. arXiv:2103.02477.

Peters, J., Janzing, D., & Schölkopf, B. (2017). *Elements of causal inference: foundations and learning algorithms* (p. 288). The MIT Press.

Pearl, J., & Mackenzie, D. (2018). *The book of why: the new science of cause and effect*. Basic books.

Runge, J. (2018) Conditional independence testing based on a nearest-neighbor estimator of conditional mutual information. In: International Conference on Artificial Intelligence and Statistics. PMLR, 938–947.

Runge, J. (2020) Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets. In: Conference on Uncertainty in Artificial Intelligence. PMLR, 1388–1397.

Runge, J., Nowack, P., Kretschmer, M., Flaxman, S. and Sejdinovic, D. (2019) Detecting and quantifying causal associations in large nonlinear time series datasets. Science Advances, 5(11), eaau4996.

Runge, J. (2015). Quantifying information transfer and mediation along causal pathways in complex systems. *Physical Review E*, *92*(6), 062829.

Runge, J., Petoukhov, V., Donges, J. F., Hlinka, J., Jajcay, N., Vejmelka, M., ... & Kurths, J. (2015). Identifying causal gateways and mediators in complex spatio-temporal systems. *Nature communications*, *6*(1), 1-10.

Runge, J., Petoukhov, V., Donges, J.F., Hlinka, J., Jajcay, N., Vejmelka, M., Hartman, D., Marwan, N., Paluš, M. and Kurths, J. (2015) Identifying causal gateways and mediators in complex spatio-temporal systems. Nature communications, 6(1), 1–10.

Runge, J. (2018). Causal network reconstruction from time series: From theoretical assumptions to practical estimation. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *28*(7), 075310.

Simpson, E. H. (1951). The interpretation of interaction in contingency tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, *13*(2), 238-241.

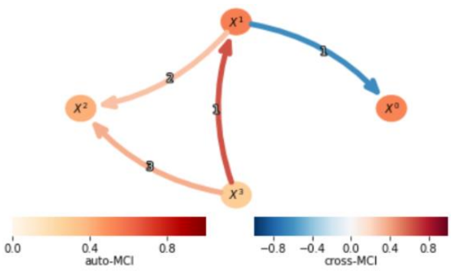
Spirtes, P., Glymour, C. N., Scheines, R., & Heckerman, D. (2000). *Causation, prediction, and search*. MIT press.

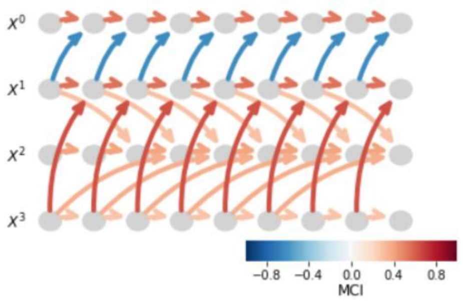
Székely, G. J., Rizzo, M. L., & Bakirov, N. K. (2007). Measuring and testing dependence by correlation of distances. *The annals of statistics*, *35*(6), 2769-2794.

Voloskov, D. and Pissarenko, D. (2020) Adaptive POD Galerkin technique for reservoir simulation and optimization. arXiv preprint arXiv:2008.03471.

7. Annexes

7.1. Annex 01 – ParCorr Tigramite Example



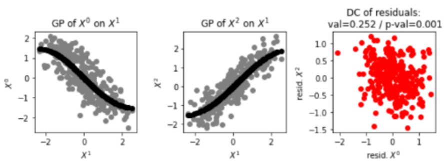


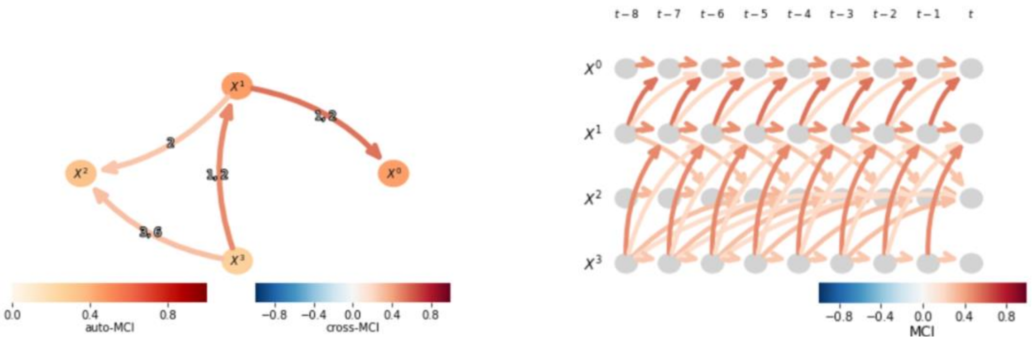
```
def jtg_ParCorr_eg():
  '''ParCorr example given'''
  np.random.seed(42)      # Fix random seed
  links_coeffs = {0: [((0, -1), 0.7), ((1, -1), -0.8)],
                  1: [((1, -1), 0.8), ((3, -1), 0.8)],
                  2: [((2, -1), 0.5), ((1, -2), 0.5), ((3, -3), 0.6)],
                  3: [((3, -1), 0.4)],
                  }
  T = 300      # time series length
  data, true_parents_neighbors = pp.var_process(links_coeffs, T=T)
  T, N = data.shape

  # Initialize dataframe object, specify time axis and variable names
  var_names = [r'$X^0$', r'$X^1$', r'$X^2$', r'$X^3$']
  dataframe = pp.DataFrame(data,
                           datatime = np.arange(len(data)),
                           var_names=var_names)
  tp.plot_timeseries(dataframe); plt.show()

  parcorr = ParCorr(significance='analytic')
  pcmci = PCMCI(
      dataframe=dataframe,
      cond_ind_test=parcorr,
      verbosity=1)

  correlations = pcmci.get_lagged_dependencies(tau_max=20, val_only=True)['val_matr
ix']
  lag_func_matrix = tp.plot_lagfuncs(val_matrix=correlations, setup_args={'var_name
s':var_names,
                                      'x_base':5, 'y_base':.5}); plt.show()

  pcmci.verbosity = 1
  results = pcmci.run_pcmci(tau_max=8, pc_alpha=1)

  q_matrix = pcmci.get_corrected_pvalues(p_matrix=results['p_matrix'],
                                          tau_max=8,
                                          fdr_method='fdr_bh')

  link_matrix = pcmci.return_significant_links(pq_matrix=q_matrix,
                                                val_matrix=results['val_matrix'],
                                                alpha_level=0.05)['link_matrix']

  tp.plot_graph(val_matrix=results['val_matrix'],link_matrix=link_matrix,
      var_names=var_names, link_colorbar_label='cross-MCI',
      node_colorbar_label='auto-MCI',); plt.show()
  tp.plot_time_series_graph(figsize=(6, 4),val_matrix=results['val_matrix'],
      link_matrix=link_matrix, var_names=var_names,
      link_colorbar_label='MCI',); plt.show()
```

7.2. Annex 02 – ParCorr Simulated Data

```
def jtg_ParCorr(chosen_data_set):
  '''Insert chosen_data_set. ParCorr test with synthetic reservoir data'''
  chosen_data_set = chosen_data_set
  T = (len(chosen_data_set['time']))
  N = 7
  datax = np.zeros((T,N))
  datax[:,0] = chosen_data_set['rate'][0]['water_rate']
  datax[:,1] = chosen_data_set['rate'][1]['water_rate']
  datax[:,2] = chosen_data_set['rate'][2]['water_rate']
  datax[:,3] = chosen_data_set['rate'][3]['water_rate']
  datax[:,4] = (chosen_data_set['rate'][4]['water_rate']) + (chosen_data_set['rate'][4]['oil_r
ate'])
  datax[:,5] = chosen_data_set['rate'][4]['water_rate']
  datax[:,6] = chosen_data_set['rate'][4]['oil_rate']
  var_names = [r'I0', r'I1', r'I2', r'I3', r'CP', r'Pw', r'Po']
  dataframe = pp.DataFrame(datax,datatime = np.arange(len(datax)),var_names=var_names)
  tp.plot_timeseries(dataframe, figsize=(10,4)); plt.show()

  #---- Selected links
  print("Type the number of Tau_min to be used for calculations:")
  jtau_min = int(input())
  print('Type the number of Tau_max  to be used for calculations:')
  jtau_max = int(input())

  tau_min = jtau_min
  tau_max = jtau_max
  selected_linksx = {0:[], 1:[], 2:[], 3:[], 4:[], 5:[], 6:[]}

  selected_linksx[4] = [(var, -
lag) for var in range(4) for lag in range(tau_min, tau_max + 1)]
  selected_linksx[5] = [(4, -lag) for lag in range(tau_min, tau_max + 1)]
  selected_linksx[6] = [(4, -lag) for lag in range(tau_min, tau_max + 1)]

  print("Selected Causal Links from var_3 (Injector 0)", selected_linksx[0])
  print("Selected Causal Links from var_3 (Injector 1)", selected_linksx[1])
  print("Selected Causal Links from var_3 (Injector 2)", selected_linksx[2])
  print("Selected Causal Links from var_3 (Injector 3)", selected_linksx[3])
  print("Selected Causal Links from var_4 (Cumulative Production)", selected_linksx[4])
  print("Selected Causal Links from var_5 (Prod of Water)", selected_linksx[5])
  print("Selected Causal Links from var_6 (Prod of Oil)", selected_linksx[6])

  # ---- Method of Calculation
  parcorr = ParCorr(significance='analytic')
  pcmci = PCMCI(
      dataframe=dataframe,
      cond_ind_test=parcorr,
      verbosity=1)

  # ---- Plotting Correlation Graph

  print("Type the number of Tau_min to be plot on correlation graph:")
  jtau_min_g = int(input())
  print('Type the number of Tau_max  to be plot on correlation graph:')
  jtau_max_g = int(input())

  correlations = pcmci.get_lagged_dependencies(selected_links=selected_linksx, tau_min = jtau_
min_g,
                                              tau_max = jtau_max_g,
                                              val_only=True)['val_matrix']

  lag_func_matrix = tp.plot_lagfuncs(val_matrix=correlations,
                                    setup_args={'var_names':var_names,
                                                'figsize':(28, 16),
                                                'x_base':5,
                                                'y_base':1}); plt.show()

  print('Type pc_alpha value (Significance level in algorithm.')
  print('Preferably, choose values in this range')
  print('if not sure, choose 1.')
  print('([0.001, 0.005, 0.01, 0.025, 0.05, 0.1,..., 1, 2])):')
  jpc_alpha = int(input())
  print('pc_alpha value = ',jpc_alpha)

  print('Alpha_level (Significance level in algorithm.')
  print('diff from PC_alpha.Default 0.05')
  print('if not sure, choose 0.05, or 0.1, or 1.')
  print('Look documentation for more details')
  jalpha_level = int(input())
  print('pc_alpha value = ',jalpha_level)

  pcmci.verbosity = 1
  results = pcmci.run_pcmci(tau_min = jtau_min, tau_max=jtau_max, pc_alpha=jpc_alpha,selected_
links=selected_linksx)

  q_matrix = pcmci.get_corrected_pvalues(p_matrix=results['p_matrix'],tau_min = jtau_min, tau_
max=jtau_max, fdr_method='fdr_bh')

  link_matrix = pcmci.return_significant_links(pq_matrix=q_matrix, val_matrix=results['val_mat
rix'], alpha_level=0.05)['link_matrix']

  node_posx = {'x':np.array((0,0.2,0.4,0.6,0.3,0.1,0.5)), # Var Y position
              'y':np.array((1,  1,1,1,0.5,0,0))} # Var Y position

  #link_width_6v = np.ones((results['val_matrix'].shape))
  #link_width1.shape = (N_vars, N_Vars, Tau_Max+1)
  #link_width_6v[0,:,:] = 1 #   arrows width from Var_names[0] | From [0] variable
  #link_width_6v[1,:,:] = 2 #   arrows width from Var_names[1] | From [1] variable
  #link_width_6v[2,:,:] = 1 #   arrows width from Var_names[2] | From [2] variable
  #link_width_6v[3,:,:] = 0.5 # arrows width from Var_names[3] | From [3] variable
  #link_width_6v[4,:,:] = 0.5 # arrows width from Var_names[4] | From [4] variable
  #link_width_6v[5,:,:] = 0.5 # arrows width from Var_names[4] | From [5] variable

  tp.plot_graph(
      link_matrix=link_matrix,
      val_matrix=results['val_matrix'],
      var_names=var_names,
      figsize=(8,8),
      link_colorbar_label='MCI',
      node_colorbar_label='auto-MCI',
      node_pos=node_posx,
      arrow_linewidth=10.0,
      vmin_edges=-1,
      vmax_edges=1.0,
      edge_ticks=0.4, #0.4
      cmap_edges='seismic',#'RdBu_r',
      vmin_nodes=0,
      vmax_nodes=1.0,
      node_ticks=0.2, #0.4
      cmap_nodes='YlGn',#OrRd
      node_size=0.15, #0.3
      arrowhead_size=20,
      curved_radius=0.3,
      label_fontsize=16, #10
      alpha=1.0,
      node_label_size=18, #10
      link_label_fontsize=10,
      network_lower_bound=0.2, #0.2
      inner_edge_style='dashed'
      ); plt.show()

  tp.plot_time_series_graph(
      link_matrix=link_matrix, val_matrix=results['val_matrix'],
      var_names=var_names,
      figsize=(18,8),
      link_colorbar_label='MCI',
      save_name=None,
      link_width=None,
      link_attribute=None,
      arrow_linewidth=8,
      vmin_edges=-1, #-1
      vmax_edges=1.0,
      edge_ticks=0.2, #0.4
      cmap_edges='seismic', #'RdBu_r'
      order=None,
      node_size=0.05, #0.1
      node_aspect=None, #None
      arrowhead_size=10, #20
      curved_radius=0.3, #0.2
      label_fontsize=18,
      alpha=1, #1.0
      node_label_size=2,
      label_space_left=0.1, #0.1
      label_space_top=0.1, #0.0
      network_lower_bound=0.2, #0.2
      inner_edge_style='dashed'
      ); plt.show()
```

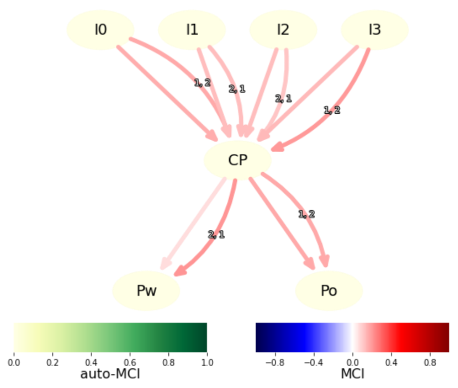
7.3. Annex 03 – GPDC Tigramite Example

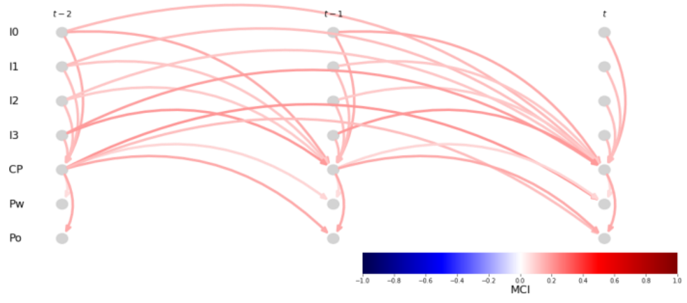




```
def jtg_GPDC_eg():
  '''GPDC example given'''
  np.random.seed(42)     # Fix random seed
  links_coeffs = {0: [((0, -1), 0.7), ((1, -1), -0.8)],
                  1: [((1, -1), 0.8), ((3, -1), 0.8)],
                  2: [((2, -1), 0.5), ((1, -2), 0.5), ((3, -3), 0.6)],
                  3: [((3, -1), 0.4)],
                  }
  T = 300     # time series length
  data, true_parents_neighbors = pp.var_process(links_coeffs, T=T)
  T, N = data.shape

# Initialize dataframe object, specify time axis and variable names
  var_names = [r'$X^0$', r'$X^1$', r'$X^2$', r'$X^3$']
  dataframe = pp.DataFrame(data,
                         datatime = np.arange(len(data)),
                         var_names=var_names)
  tp.plot_timeseries(dataframe); plt.show()

  gpdc = GPDC(significance='analytic', gp_params=None)
  pcmci = PCMCI(
      dataframe=dataframe,
      cond_ind_test=gpdc,
      verbosity=0)

  correlations = pcmci.get_lagged_dependencies(tau_max=20, val_only=True)['val_matrix']
  lag_func_matrix = tp.plot_lagfuncs(val_matrix=correlations, setup_args={'var_names':var_name
s,'x_base':5, 'y_base':.5}); plt.show()

  pcmci.verbosity = 1
  results = pcmci.run_pcmci(tau_max=8, pc_alpha=1)

  q_matrix = pcmci.get_corrected_pvalues(p_matrix=results['p_matrix'],
                                         tau_max=8, fdr_method='fdr_bh')
  link_matrix = pcmci.return_significant_links(pq_matrix=q_matrix,
                                               val_matrix=results['val_matrix'],
                                               alpha_level=0.05)['link_matrix']

# Scatter plot

  array, dymmy, dummy = gpdc._get_array(X=[(0, -1)], Y=[(2, 0)], Z=[(1, -2)], tau_max=2)
  x, meanx = gpdc._get_single_residuals(array, target_var=0, return_means=True)
  y, meany = gpdc._get_single_residuals(array, target_var=1, return_means=True)

  fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(8,3))
  axes[0].scatter(array[2], array[0], color='grey')
  axes[0].scatter(array[2], meanx, color='black')
  axes[0].set_title("GP of %s on %s" % (var_names[0], var_names[1]) )
  axes[0].set_xlabel(var_names[1]); axes[0].set_ylabel(var_names[0])
  axes[1].scatter(array[2], array[1], color='grey')
  axes[1].scatter(array[2], meany, color='black')
  axes[1].set_title("GP of %s on %s" % (var_names[2], var_names[1]) )
  axes[1].set_xlabel(var_names[1]); axes[1].set_ylabel(var_names[2])
  axes[2].scatter(x, y, color='red')
  axes[2].set_title("DC of residuals:" "\n val=%.3f / p-val=%.3f" % (gpdc.run_test(
              X=[(0, -1)], Y=[(2, 0)], Z=[(1, -2)], tau_max=2)) )
  axes[2].set_xlabel("resid. "+var_names[0]); axes[2].set_ylabel("resid. "+var_names[2])
  plt.tight_layout()

  tp.plot_graph(val_matrix=results['val_matrix'],
      link_matrix=link_matrix, var_names=var_names,
      link_colorbar_label='cross-MCI', node_colorbar_label='auto-MCI',
      ); plt.show()

  tp.plot_time_series_graph(figsize=(6, 4),val_matrix=results['val_matrix'],
      link_matrix=link_matrix, var_names=var_names,
      link_colorbar_label='MCI'); plt.show()
```

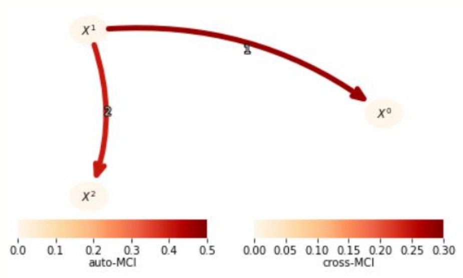
7.4. Annex 04 – GPDC Simulated Data

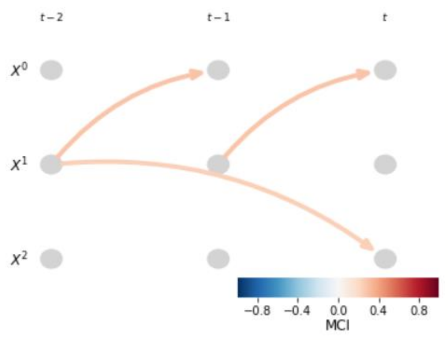




```
def jtg_GPDC(chosen_data_set):
  '''Insert chosen_data_set. GPDC test with synthetic reservoir data'''
  chosen_data_set = chosen_data_set
  T = (len(chosen_data_set['time']))
  N = 7
  datax = np.zeros((T,N))
  datax[:,0] = chosen_data_set['rate'][0]['water_rate']
  datax[:,1] = chosen_data_set['rate'][1]['water_rate']
  datax[:,2] = chosen_data_set['rate'][2]['water_rate']
  datax[:,3] = chosen_data_set['rate'][3]['water_rate']
  datax[:,4] = (chosen_data_set['rate'][4]['water_rate']) + (chosen_data_set['rate'][4]['oil_r
ate'])
  datax[:,5] = chosen_data_set['rate'][4]['water_rate']
  datax[:,6] = chosen_data_set['rate'][4]['oil_rate']
  var_names = [r'I0', r'I1', r'I2', r'I3', r'CP', r'Pw', r'Po']
  dataframe = pp.DataFrame(datax,datatime = np.arange(len(datax)),var_names=var_names)
  tp.plot_timeseries(dataframe); plt.show()

  #---- Selected links
  print("Type the number of Tau_min to be used for calculations:")
  jtau_min = int(input())
  print('Type the number of Tau_max  to be used for calculations:')
  jtau_max = int(input())

  tau_min = jtau_min
  tau_max = jtau_max

  selected_linksx = {0:[], 1:[], 2:[], 3:[], 4:[], 5:[], 6:[]}

  selected_linksx[4] = [(var, -
lag) for var in range(4) for lag in range(tau_min, tau_max + 1)]
  selected_linksx[5] = [(4, -lag) for lag in range(tau_min, tau_max + 1)]
  selected_linksx[6] = [(4, -lag) for lag in range(tau_min, tau_max + 1)]

  print("Selected Causal Links from var_3 (Injector 0)", selected_linksx[0])
  print("Selected Causal Links from var_3 (Injector 1)", selected_linksx[1])
  print("Selected Causal Links from var_3 (Injector 2)", selected_linksx[2])
  print("Selected Causal Links from var_3 (Injector 3)", selected_linksx[3])
  print("Selected Causal Links from var_4 (Cumulative Production)", selected_linksx[4])
  print("Selected Causal Links from var_5 (Prod of Water)", selected_linksx[5])
  print("Selected Causal Links from var_6 (Prod of Oil)", selected_linksx[6])

# ---- Method of Calculation
  gpdc = GPDC(significance='analytic', gp_params=None)
  pcmci = PCMCI(
      dataframe=dataframe,
      cond_ind_test=gpdc,
      verbosity=0)

# ---- Plotting Correlation Graph

  correlations = pcmci.get_lagged_dependencies(selected_links=selected_linksx, tau_min=jtau_mi
n,
                                               tau_max=jtau_max, val_only=True)['val_matrix']

  lag_func_matrix = tp.plot_lagfuncs(val_matrix=correlations, setup_args={'var_names':var_name
s,
                                    'x_base':5, 'y_base':.5}); plt.show()

  pcmci.verbosity = 1
  results = pcmci.run_pcmci(selected_links=selected_linksx, tau_min=jtau_min,
                            tau_max=jtau_max, pc_alpha=1)

  q_matrix = pcmci.get_corrected_pvalues(p_matrix=results['p_matrix'],
                                         tau_min=jtau_min,
                                         tau_max=jtau_max,
                                         fdr_method='fdr_bh')

  link_matrix = pcmci.return_significant_links(pq_matrix=q_matrix,
                                               val_matrix=results['val_matrix'],
                                               alpha_level=1)['link_matrix']

  node_posx = {'x':np.array((0,0.2,0.4,0.6,0.3,0.1,0.5)), # Var Y position
               'y':np.array((1,  1,1,1,0.5,0,0))} # Var Y position

# --- if the link (arrows) width need to be different

  #link_width = np.ones((results['val_matrix'].shape))
  #link_width.shape = (N_vars, N_Vars, Tau_Max+1)
  #link_width[0,:,:] = 1 #   arrows width from Var_names[0] | From [0] variable
  #link_width[1,:,:] = 2 #   arrows width from Var_names[1] | From [1] variable
  #link_width[2,:,:] = 1 #   arrows width from Var_names[2] | From [2] variable
  #link_width[3,:,:] = 0.5 # arrows width from Var_names[3] | From [3] variable
  #link_width[4,:,:] = 0.5 # arrows width from Var_names[4] | From [4] variable
  #link_width[5,:,:] = 0.5 # arrows width from Var_names[4] | From [5] variable

# --- Plot DAG
  tp.plot_graph(
      link_matrix=link_matrix,
      val_matrix=results['val_matrix'],
      var_names=var_names,
      figsize=(8,8),
      link_colorbar_label='MCI',
      node_colorbar_label='auto-MCI',
      node_pos=node_posx,
      arrow_linewidth=10.0,
      vmin_edges=-1,
      vmax_edges=1.0,
      edge_ticks=0.4, #0.4
      cmap_edges='seismic',#'RdBu_r',
      vmin_nodes=0,
      vmax_nodes=1.0,
      node_ticks=0.2, #0.4
      cmap_nodes='YlGn',#OrRd
      node_size=0.15, #0.3
      arrowhead_size=20,
      curved_radius=0.3,
      label_fontsize=16, #10
      alpha=1.0,
      node_label_size=18, #10
      link_label_fontsize=10,
      network_lower_bound=0.2, #0.2
      inner_edge_style='dashed'
      ); plt.show()

# --- Plot time-series graph
  tp.plot_time_series_graph(
      link_matrix=link_matrix,
      val_matrix=results['val_matrix'],
      var_names=var_names,
      figsize=(18,8),
      link_colorbar_label='MCI',
      save_name=None,
      link_width=None,
      link_attribute=None,
      arrow_linewidth=8,
      vmin_edges=-1, #-1
      vmax_edges=1.0,
      edge_ticks=0.2, #0.4
      cmap_edges='seismic', #'RdBu_r'
      order=None,
      node_size=0.05, #0.1
      node_aspect=None, #None
      arrowhead_size=10, #20
      curved_radius=0.3, #0.2
      label_fontsize=18,
      alpha=1, #1.0
      node_label_size=2,
      label_space_left=0.1, #0.1
      label_space_top=0.1, #0.0
      network_lower_bound=0.2, #0.2
      inner_edge_style='dashed'
      ); plt.show()
```

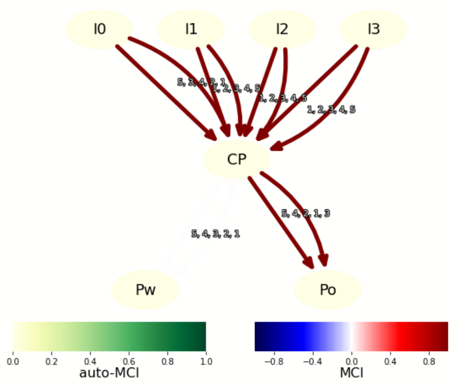
7.5. Annex 05 – CMIknn Tigramite Example

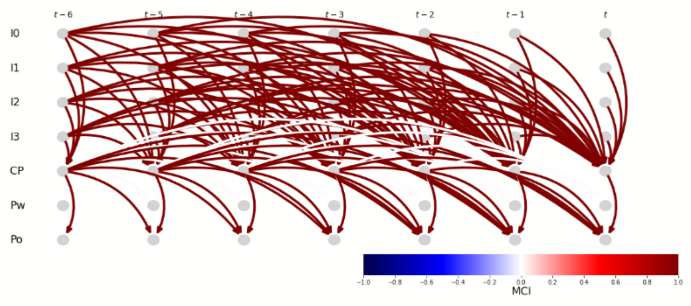




```
def jtg_CMIknn_eg():
  '''CMIknn example given'''
  np.random.seed(42)     # Fix random seed
  links_coeffs = {0: [((0, -1), 0.7), ((1, -1), -0.8)],
                  1: [((1, -1), 0.8), ((3, -1), 0.8)],
                  2: [((2, -1), 0.5), ((1, -2), 0.5), ((3, -3), 0.6)],
                  3: [((3, -1), 0.4)],
                  }
  T = 300     # time series length
  data, true_parents_neighbors = pp.var_process(links_coeffs, T=T)
  T, N = data.shape

# Initialize dataframe object, specify time axis and variable names
  var_names = [r'$X^0$', r'$X^1$', r'$X^2$', r'$X^3$']
  dataframe = pp.DataFrame(data,
                           datatime = np.arange(len(data)),
                           var_names=var_names)
  tp.plot_timeseries(dataframe); plt.show()

  cmi_knn = CMIknn(significance='shuffle_test', knn=0.1, shuffle_neighbors=10, transform='rank
s')
  pcmci_cmi_knn = PCMCI(dataframe=dataframe, cond_ind_test=cmi_knn, verbosity=0)

  correlations_CMIknn = pcmci_cmi_knn.get_lagged_dependencies(tau_max=10,tau_min=0, val_only=T
rue)['val_matrix']
  lag_func_matrix_CMIknn =tp.plot_lagfuncs(val_matrix=correlations_CMIknn,
                                           setup_args={'var_names':var_names,'figsize':(16,12)
,'minimum':.0,'plot_gridlines':True,'x_base':1, 'y_base':.25});

  link_matrix = pcmci_cmi_knn.return_significant_links(pq_matrix=results['p_matrix'],
                          val_matrix=results['val_matrix'], alpha_level=0.01)['link_matrix']

  tp.plot_graph(val_matrix=results['val_matrix'],link_matrix=link_matrix,
      var_names=var_names,link_colorbar_label='cross-MCI',
      node_colorbar_label='auto-MCI',vmin_edges=0.,vmax_edges = 0.3,
      edge_ticks=0.05, cmap_edges='OrRd', vmin_nodes=0,
      vmax_nodes=.5, node_ticks=.1,
      cmap_nodes='OrRd',); plt.show()

  tp.plot_time_series_graph(figsize=(6, 4),val_matrix=results['val_matrix'],
      link_matrix=link_matrix, var_names=var_names,link_colorbar_label='MCI'); plt.show()
```

7.6. Annex 06 – CMIknn Simulated Data





```
def jtg_CMIknn(chosen_data_set):
  '''Insert chosen_data_set. CMIknn test with synthetic reservoir data'''
  chosen_data_set = chosen_data_set
  T = (len(chosen_data_set['time']))
  N = 7
  datax = np.zeros((T,N))
  datax[:,0] = chosen_data_set['rate'][0]['water_rate']
  datax[:,1] = chosen_data_set['rate'][1]['water_rate']
  datax[:,2] = chosen_data_set['rate'][2]['water_rate']
  datax[:,3] = chosen_data_set['rate'][3]['water_rate']
  datax[:,4] = (chosen_data_set['rate'][4]['water_rate']) + (chosen_data_set['rate'][4]['oil_r
ate'])
  datax[:,5] = chosen_data_set['rate'][4]['water_rate']
  datax[:,6] = chosen_data_set['rate'][4]['oil_rate']
  var_names = [r'I0', r'I1', r'I2', r'I3', r'CP', r'Pw', r'Po']
  dataframe = pp.DataFrame(datax,datatime = np.arange(len(datax)),var_names=var_names)
  tp.plot_timeseries(dataframe, figsize=(10,4)); plt.show()
  #---- Selected links
  print("Type the number of Tau_min to be used for calculations:")
  jtau_min = int(input())
  print('Type the number of Tau_max  to be used for calculations:')
  jtau_max = int(input())

  tau_min = jtau_min
  tau_max = jtau_max
  selected_linksx = {0:[], 1:[], 2:[], 3:[], 4:[], 5:[], 6:[]}

  selected_linksx[4] = [(var, -
lag) for var in range(4) for lag in range(tau_min, tau_max + 1)]
  selected_linksx[5] = [(4, -lag) for lag in range(tau_min, tau_max + 1)]
  selected_linksx[6] = [(4, -lag) for lag in range(tau_min, tau_max + 1)]

  print("Selected Causal Links from var_3 (Injector 0)", selected_linksx[0])
  print("Selected Causal Links from var_3 (Injector 1)", selected_linksx[1])
  print("Selected Causal Links from var_3 (Injector 2)", selected_linksx[2])
  print("Selected Causal Links from var_3 (Injector 3)", selected_linksx[3])
  print("Selected Causal Links from var_4 (Cumulative Production)", selected_linksx[4])
  print("Selected Causal Links from var_5 (Prod of Water)", selected_linksx[5])
  print("Selected Causal Links from var_6 (Prod of Oil)", selected_linksx[6])

  # ---- Method of Calculation
  cmi_knn = CMIknn(significance='shuffle_test', knn=0.1, shuffle_neighbors=5, transform='ranks
')
  pcmci_cmi_knn = PCMCI(
      dataframe=dataframe,
      cond_ind_test=cmi_knn,
      verbosity=2)
  results = pcmci_cmi_knn.run_pcmci(selected_links=selected_linksx, tau_max=6, pc_alpha=0.05)
  pcmci_cmi_knn.print_significant_links(
          p_matrix = results['p_matrix'],
          val_matrix = results['val_matrix'],
          alpha_level = 0.01)

  # ---- Plotting Correlation Graph

  print("Type the number of Tau_min to be plot on correlation graph:")
  jtau_min_g = int(input())
  print('Type the number of Tau_max  to be plot on correlation graph:')
  tau_max_g = int(input())

  link_matrix = pcmci_cmi_knn.return_significant_links(pq_matrix=results['p_matrix'],
                        val_matrix=results['val_matrix'], alpha_level=0.01)['link_matrix']

  print('Type pc_alpha value (Significance level in algorithm.')
  print('Preferably, choose values in this range')
  print('if not sure, choose 1.')
  print('([0.001, 0.005, 0.01, 0.025, 0.05, 0.1,..., 1, 2])):')
  jpc_alpha = int(input())
  print('pc_alpha value = ',jpc_alpha)
  print('Alpha_level (Significance level in algorithm.')
  print('diff from PC_alpha.Default 0.05')
  print('if not sure, choose 0.05, or 0.1, or 1.')
  print('Look documentation for more details')
  jalpha_level = int(input())
  print('pc_alpha value = ',jalpha_level)

  node_posx = {'x':np.array((0,0.2,0.4,0.6,0.3,0.1,0.5)), # Var Y position
               'y':np.array((1,  1,1,1,0.5,0,0))} # Var Y position

# --- if the link (arrows) width need to be different

  #link_width = np.ones((results['val_matrix'].shape))
  #link_width.shape = (N_vars, N_Vars, Tau_Max+1)
  #link_width[0,:,:] = 1 #   arrows width from Var_names[0] | From [0] variable
  #link_width[1,:,:] = 2 #   arrows width from Var_names[1] | From [1] variable
  #link_width[2,:,:] = 1 #   arrows width from Var_names[2] | From [2] variable
  #link_width[3,:,:] = 0.5 # arrows width from Var_names[3] | From [3] variable
  #link_width[4,:,:] = 0.5 # arrows width from Var_names[4] | From [4] variable
  #link_width[5,:,:] = 0.5 # arrows width from Var_names[4] | From [5] variable

# --- Plot DAG
  tp.plot_graph(
      link_matrix=link_matrix,
      val_matrix=results['val_matrix'],
      var_names=var_names,
      figsize=(8,8),
      link_colorbar_label='MCI',
      node_colorbar_label='auto-MCI',
      node_pos=node_posx,
      arrow_linewidth=10.0,
      vmin_edges=-1,
      vmax_edges=1.0,
      edge_ticks=0.4, #0.4
      cmap_edges='seismic',#'RdBu_r',
      vmin_nodes=0,
      vmax_nodes=1.0,
      node_ticks=0.2, #0.4
      cmap_nodes='YlGn',#OrRd
      node_size=0.15, #0.3
      arrowhead_size=20,
      curved_radius=0.3,
      label_fontsize=16, #10
      alpha=1.0,
      node_label_size=18, #10
      link_label_fontsize=10,
      network_lower_bound=0.2, #0.2
      inner_edge_style='dashed'
      ); plt.show()

# --- Plot time-series graph
  tp.plot_time_series_graph(
      link_matrix=link_matrix,
      val_matrix=results['val_matrix'],
      var_names=var_names,
      figsize=(18,8),
      link_colorbar_label='MCI',
      link_attribute=None,
      arrow_linewidth=8,
      vmin_edges=-1, #-1
      vmax_edges=1.0,
      edge_ticks=0.2, #0.4
      cmap_edges='seismic', #'RdBu_r'
      node_size=0.05, #0.1
      node_aspect=None, #None
      arrowhead_size=10, #20
      curved_radius=0.3, #0.2
      label_fontsize=18,
      alpha=1, #1.0
      node_label_size=2,
      label_space_left=0.1, #0.1
      label_space_top=0.1, #0.0
      network_lower_bound=0.2, #0.2
      inner_edge_style='dashed'
      ); plt.show()
```

7.7. Annex 07 – Parallel computing using mpi4py

Author: Jakob Runge.

License: GNU General Public License v3.0

```
"""
        Tigramite causal discovery for time series: Parallization script implementing
        the PCMCI method based on mpi4py.
        Parallelization is done across variables j for both the PC condition-selection
        step and the MCI step.
        """

        from mpi4py import MPI
        import numpy
        import os, sys, pickle


        from tigramite import data_processing as pp
        from tigramite.pcmci import PCMCI
        from tigramite.independence_tests import ParCorr, GPDC, CMIknn, CMIsymb

        # Default communicator
        COMM = MPI.COMM_WORLD

        def split(container, count):
```

```
    """
    Simple function splitting a the range of selected variables (or range(N))
    into equal length chunks. Order is not preserved.
    """
    return [container[_i::count] for _i in range(count)]

def run_pc_stable_parallel(j):
    """Wrapper around PCMCI.run_pc_stable estimating the parents for a single
    variable j.
    Parameters
    ----------
    j : int
        Variable index.
    Returns
    -------
    j, pcmci_of_j, parents_of_j : tuple
        Variable index, PCMCI object, and parents of j
    """
    # CondIndTest is initialized globally below
    # Further parameters of PCMCI as described in the documentation can be
    # supplied here:
    pcmci_of_j = PCMCI(
        dataframe=dataframe,
        cond_ind_test=cond_ind_test,
        selected_variables=[j],
        verbosity=verbosity)


    # Run PC condition-selection algorithm. Also here further parameters can be
    # specified:
    parents_of_j = pcmci_of_j.run_pc_stable(
                      selected_links=selected_links,
                      tau_max=tau_max,
                      pc_alpha=pc_alpha,
            )


    # We return also the PCMCI object because it may contain pre-computed
    # results can be re-used in the MCI step (such as residuals or null
    # distributions)
    return j, pcmci_of_j, parents_of_j

def run_mci_parallel(j, pcmci_of_j, all_parents):
    """Wrapper around PCMCI.run_mci step.
    Parameters
    ----------
    j : int
        Variable index.
    pcmci_of_j : object
        PCMCI object for variable j. This may contain pre-computed results
        (such as residuals or null distributions).
    all_parents : dict
        Dictionary of parents for all variables. Needed for MCI independence
        tests.
    Returns
    -------
    j, results_in_j : tuple
        Variable index and results dictionary containing val_matrix, p_matrix,
        and optionally conf_matrix with non-zero entries only for
        matrix[:,j,:].
    """
    results_in_j = pcmci_of_j.run_mci(
                selected_links=selected_links,
                tau_min=tau_min,
                tau_max=tau_max,
                parents=all_parents,
                max_conds_px=max_conds_px,
            )


    return j, results_in_j

# Example data, here the real dataset can be loaded as a numpy array of shape
# (T, N)
```

```
numpy.random.seed(42)     # Fix random seed
links_coeffs = {0: [((0, -1), 0.7)],
                1: [((1, -1), 0.8), ((0, -1), 0.8)],
                2: [((2, -1), 0.5), ((1, -2), 0.5)],
                }


T = 500     # time series length
data, true_parents_neighbors = pp.var_process(links_coeffs, T=T)
T, N = data.shape


# Optionally specify variable names
var_names = [r'$X^0$', r'$X^1$', r'$X^2$', r'$X^3$']

# Initialize dataframe object
dataframe = pp.DataFrame(data, var_names=var_names)

# Significance level in condition-selection step. If a list of levels is is
# provided or pc_alpha=None, the optimal pc_alpha is automatically chosen via
# model-selection.
pc_alpha = 0.2  # [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]
selected_variables = range(N)  #[2] # [2]  # [2]

# Maximum time lag
tau_max = 3
# Optional minimum time lag in MCI step (in PC-step this is 1)

tau_min = 0
# Maximum cardinality of conditions in PC condition-selection step. The
# recommended default choice is None to leave it unrestricted.
max_conds_dim = None


# Maximum number of parents of X to condition on in MCI step, leave this to None
# to condition on all estimated parents.
max_conds_px = None


# Selected links may be used to restricted estimation to given links.
selected_links = None


# Alpha level for MCI tests (just used for printing since all p-values are
# stored anyway)
alpha_level = 0.05


# Verbosity level. Note that slaves will ouput on top of each other.
verbosity = 0


# Chosen conditional independence test
cond_ind_test = ParCorr()  #confidence='analytic')


# Store results in file
file_name = os.path.expanduser('~') + '/test_results.dat'

#
#  Start of the script
#
if COMM.rank == 0:
    # Only the master node (rank=0) runs this
    if verbosity > -1:
        print("\n##\n## Running Parallelized Tigramite PC algorithm\n##"
              "\n\nParameters:")
        print("\nindependence test = %s" % cond_ind_test.measure
              + "\ntau_min = %d" % tau_min
              + "\ntau_max = %d" % tau_max
              + "\npc_alpha = %s" % pc_alpha
              + "\nmax_conds_dim = %s" % max_conds_dim)
        print("\n")
    # Split selected_variables into however many cores are available.
    splitted_jobs = split(selected_variables, COMM.size)
    if verbosity > -1:
        print("Splitted selected_variables = "), splitted_jobs
else:
    splitted_jobs = None

##
##  PC algo condition-selection step
##
# Scatter jobs across cores.
scattered_jobs = COMM.scatter(splitted_jobs, root=0)


# Now each rank just does its jobs and collects everything in a results list.
results = []
for j in scattered_jobs:
    # Estimate conditions
    (j, pcmci_of_j, parents_of_j) = run_pc_stable_parallel(j)


    results.append((j, pcmci_of_j, parents_of_j))


# Gather results on rank 0.
results = MPI.COMM_WORLD.gather(results, root=0)

if COMM.rank == 0:
    # Collect all results in dictionaries and send results to workers
    all_parents = {}
    pcmci_objects = {}
    for res in results:
        for (j, pcmci_of_j, parents_of_j) in res:
            all_parents[j] = parents_of_j[j]
            pcmci_objects[j] = pcmci_of_j


    if verbosity > -1:
        print("\n\n## Resulting condition sets:")
        for j in [var for var in all_parents.keys()]:
            pcmci_objects[j]._print_parents_single(j, all_parents[j],
                                        pcmci_objects[j].val_min[j],
                                        pcmci_objects[j].p_max[j])


    if verbosity > -1:
        print("\n##\n## Running Parallelized Tigramite MCI algorithm\n##"
              "\n\nParameters:")


        print("\nindependence test = %s" % cond_ind_test.measure
              + "\ntau_min = %d" % tau_min
              + "\ntau_max = %d" % tau_max
              + "\nmax_conds_px = %s" % max_conds_px)

        print("Master node: Sending all_parents and pcmci_objects to workers.")

    for i in range(1, COMM.size):
        COMM.send((all_parents, pcmci_objects), dest=i)


else:
    if verbosity > -1:
        print("Slave node %d: Receiving all_parents and pcmci_objects..."
              "" % COMM.rank)
    (all_parents, pcmci_objects) = COMM.recv(source=0)

##
##   MCI step
##
# Scatter jobs again across cores.
scattered_jobs = COMM.scatter(splitted_jobs, root=0)
```

```
# Now each rank just does its jobs and collects everything in a results list.
results = []
for j in scattered_jobs:
    (j, results_in_j) = run_mci_parallel(j, pcmci_objects[j], all_parents)
    results.append((j, results_in_j))


# Gather results on rank 0.
results = MPI.COMM_WORLD.gather(results, root=0)

if COMM.rank == 0:
    # Collect all results in dictionaries
    #
    if verbosity > -1:
        print("\nCollecting results...")
    all_results = {}
    for res in results:
        for (j, results_in_j) in res:
            for key in results_in_j.keys():
                if results_in_j[key] is None:
                    all_results[key] = None
                else:
                    if key not in all_results.keys():
                        if key == 'p_matrix':
                            all_results[key] = numpy.ones(results_in_j[key].shape)
                        else:
                            all_results[key] = numpy.zeros(results_in_j[key].shape)
                        all_results[key][:,j,:] =  results_in_j[key][:,j,:]
                    else:
                        all_results[key][:,j,:] =  results_in_j[key][:,j,:]




    p_matrix=all_results['p_matrix']
    val_matrix=all_results['val_matrix']
    conf_matrix=all_results['conf_matrix']


    sig_links = (p_matrix <= alpha_level)


    if verbosity > -1:
        print("\n## Significant links at alpha = %s:" % alpha_level)
        for j in selected_variables:


            links = dict([((p[0], -p[1] ), numpy.abs(val_matrix[p[0],
                            j, abs(p[1])]))
                          for p in zip(*numpy.where(sig_links[:, j, :]))])


            # Sort by value
            sorted_links = sorted(links, key=links.get, reverse=True)


            n_links = len(links)


            string = ""
            string = ("\n    Variable %s has %d "
                      "link(s):" % (var_names[j], n_links))
            for p in sorted_links:
                string += ("\n        (%s %d): pval = %.5f" %
                           (var_names[p[0]], p[1],
                            p_matrix[p[0], j, abs(p[1])]))


                string += " | val = %.3f" % (
                    val_matrix[p[0], j, abs(p[1])])


                if conf_matrix is not None:
```

```
                string += " | conf = (%.3f, %.3f)" % (
                    conf_matrix[p[0], j, abs(p[1])][0],
                    conf_matrix[p[0], j, abs(p[1])][1])


        print (string)
if verbosity > -1:
    print("Pickling to ", file_name)
file = open(file_name, 'wb')
pickle.dump(all_results, file, protocol=-1)
file.close()
# PCMCI._print_significant_links(
#        p_matrix=all_results['p_matrix'],
#        val_matrix=all_results['val_matrix'],
#        alpha_level=alpha_level,
#        conf_matrix=all_results['conf_matrix'])
```