**Beyond The Basics! Level Up Your Causal Discovery Skills in Python (2023)**

**…and unlock the potential of The Best & Most Underrated Causal Discovery package in Python!**



Image by [Pixabay at Pexels](https://www.pexels.com/@pixabay/) (<https://www.pexels.com/photo/purple-leaf-459301/>)

In this post you’ll learn how to perform **causal discovery from observational data** in **Python** using **my favorite,** most **complete** and **up-to-date** **causal discovery package** out there. By the end of…

**Introduction**

The recent surge in interest in causality-related topics in Python has led to a wealth of resources, making it challenging to find the ones that are aligned with your learning goals.

As I began my adventure with causality some three years ago, I spent countless hours scouring the internet for valuable resources and robust libraries, determined to find the tools I needed to succeed and move forward.

This blog is a part of the **series** where I share practical tips on [**learning**](https://aleksander-molak.medium.com/yes-six-causality-books-that-will-get-you-from-zero-to-advanced-2023-f4d08718a2dd) and [**implementing**](https://towardsdatascience.com/causal-kung-fu-in-python-3-basic-techniques-to-jump-start-your-causal-inference-journey-tonight-ae09181704f7) causal models in Python.

If you’re interested in modern causality, I’ve distilled **over three years** of hard-won knowledge and experience into an accessible guide that will help you master causality in a fraction of the time it would take to do all the research and coding yourself. Check my [**upcoming causal book**](https://causalpython.io/) and be sure to join our growing community at [**causalpython.io**](https://causalpython.io) for even more tips, tricks, and support!

[Links to the **notebook** and the **conda environment** file are **below**]

Let’s learn how to discover!



Image by [Alexander Ant](https://www.pexels.com/@alexant/) @ [Pexels](https://www.pexels.com/photo/abstract-background-of-bright-paints-5603660/)

**What Is Causal Discovery?**

C**asual discovery**, also known as **causal structure learning** denotes a broad set of methods aiming at retrieving information about causal mechanisms from observational or interventional data. In other words, causal discovery algorithms try to decode the **causal structure** of the **data generating process**, using the data generated by this process.

Some of these algorithms allow us to easily incorporate prior knowledge (also called *expert knowledge*) in the form of constraints. This helps narrow **down the problem space** and makes it easier for the algorithm to find a good solution.

In most cases we describe the data generating process using a [**directed acyclic graph**](https://en.wikipedia.org/wiki/Directed_acyclic_graph) (**DAG**).

**The Four Families of Causal Discovery Methods**

There are **four broad classes** of **causal discovery** algorithms:

* **Constraint-based**
* **Score-based**
* **Functional**
* **Other** (incl. hybrid, gradient-based and more)

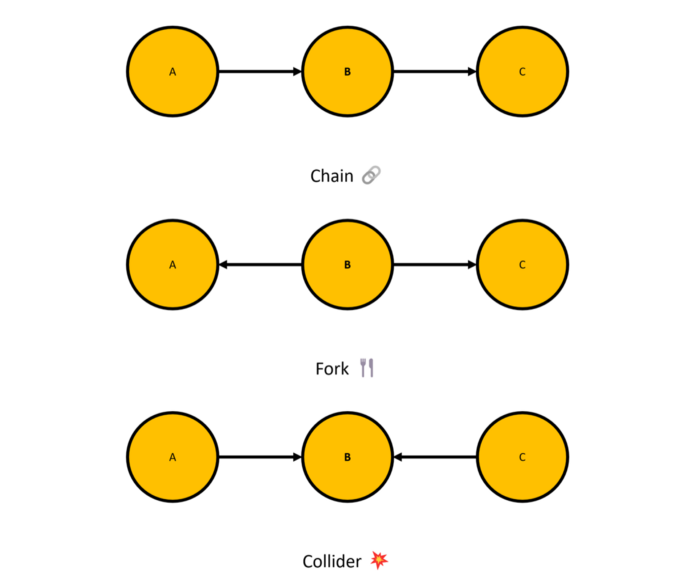
Keep in mind that this typology is not consistent in causal literature and that classes are not always mutually exclusive. That said, each brings some unique flavor.

Let’s do some tasting!

**Constraint-based methods**

Constraint-based methods (also known as *independence-based methods*) aim at decoding causal structure from the data by leveraging independence structure between the triplets of variables. Sounds dense? Let’s unpack it!

Imagine that we have a system consisting of three variables: ***A***, ***B*** and ***C***. Each variable is represented by a node in a graph and we can only have two directed edges in such a graph. Moreover, we keep these variables ordered, so that the edges can only connect nodes ***A*** and ***B*** and ***B*** and ***C***. This gives us three possible graphs. We present them in **Figure 1**.



**Figure 1.** The three basic graphical causal structures. Image by yours truly.

The arrows in the graphs above denote causal relationships between variables (we follow [**Pearl’s definition of causality**](https://causalpython.io/#define-causality) here). Each of the graphical structures presented in **Figure 1** has a specific name. From the top to bottom, they are:

* **Chain**
* **Fork**
* **Collider** (also known as *immorality* (sic!)or *v-structure*)

**Independence structures**

It turns out that under certain circumstances we can perform a mapping between the graphical structure representing the data generating process and statistical properties of the variables that are the outcome of this process. Moreover, in certain cases the mapping in the other direction — from the data to the graph — is also possible.

Among the three structures we presented in **Figure 1**, the **collider structure** has a unique property. If any three variables in your dataset have been generated from a collider-structured causal process, we can retrieve this information from the observational data using pairwise statistical independence tests¹. This means that we can recreate the graph based on the observed data itself. That’s exciting!

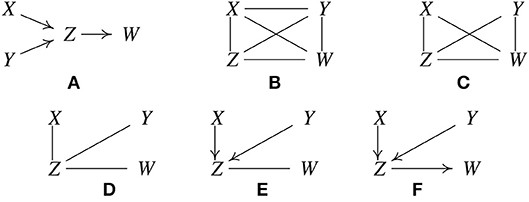
Unfortunately, things are not that smooth with forks and chains. Statistical independence structure is the same for both of these graphical structures and we cannot unambiguously map them back to the graph. Nonetheless, if we’re lucky enough, neighboring colliders can help us recover and orient the edges in fork and chain structures as well.

If you want to learn more on the properties of chains, forks and colliders, check **Brady Neal’s videos** on the topic ([1](https://www.youtube.com/watch?v=Q9CAtMpuWCA), [2](https://www.youtube.com/watch?v=5xIujBzwi7E)) or [**this part**](https://www.youtube.com/watch?v=kyRUDTexwGM&t=2063s)of my talk at **PyData Hamburg.**

**PC algorithm**

A classic example of a constratint-based algorithm is the **PC algorithm** (Sprites & Glymour, 1991). Its name comes from the first names of its creators: Peter Sprites and Clark Glymour. PC algorithm is a variant of the **IC algorithm** proposed earlier by Verma & Pearl (1990).

**Figure 2** presents a step-by-step process of the PC algorithm.



**Figure 2.** A step-by-step visualization of the PC algorithm (Glymour et al., 2019)

In order to find the ground truth (**Figure 2 A**) PC algorithm starts with a fully-connected undirected graph (**B**). Next, it removes the edges between the variables that unconditionally independent (**C**), then the ones that are conditionally independent (**D**). Finally, the algorithm finds the directed edges based on detected ***collider structures*** (**E**) and disambiguates collider-neighboring edges if possible (**F**).

Sometimes, the algorithm might not be able to orient all edges. In such a case so-called **Markov equivalence class** (**MEC**) is returned. In practice MEC means that you get a graph with some of the edges without determined direction.

A important limitation of the PC algorithm is that the results might be arbitrarily misleading if there’s hidden [confounding](https://causalpython.io/#confounding) in your data. A generalization of the PC algorithm, called FCI (Fast Causal Inference; Sprites et al., 2001) addresses this problem (at least in the asymptotic regime).

Another, more general limitation is that constraint-based algorithms like PC and IC rely on conditional independence testing, which is a difficult task in the non-parametric setting. Up to my best knowledge there’s no general non-parametric model-free solution to this problem (Azadkia et al., 2021).

Looking for **recommendations** on **causality books** to read?

**[Yes! Six Causality Books That Will Get You From Zero to Advanced (2023)](https://aleksander-molak.medium.com/yes-six-causality-books-that-will-get-you-from-zero-to-advanced-2023-f4d08718a2dd" \t "_blank)**

**[…and you can get 3 of them completely for free if you want! 🤗](https://aleksander-molak.medium.com/yes-six-causality-books-that-will-get-you-from-zero-to-advanced-2023-f4d08718a2dd" \t "_blank)**

[aleksander-molak.medium.com](https://aleksander-molak.medium.com/yes-six-causality-books-that-will-get-you-from-zero-to-advanced-2023-f4d08718a2dd" \t "_blank)



Image by [Sebastian Arie Voortman](https://www.pexels.com/@sebastian/) @ [Pexels](https://www.pexels.com/photo/two-silver-chess-pieces-on-white-surface-411207/)

**Score-based methods**

Score-based methods work by iteratively generating candidate graphs, evaluating how well each one explains the data, and selecting the best one. One well-known example of a score-based method is **Greedy Equivalence Search** (**GES**) introduced by David Maxwell Chickering (Chickering, 2003).

**GES**

The algorithm is a two-stage procedure. First, it generates the edges, then it prunes the graph.

The first stage of **GES** begins with an unconnected graph. The algorithm then iteratively adds edges, computing the score at each step². This continues until the score can no longer be increased. In the second stage, the algorithm starts to *prune* the existing edges in order to see if the score can be improved further. All these computations are performed in a greedy fashion (hence the name).

Similarly to the PC algorithm, GES is sensitive to hidden confounding. It might be also unable to orient all edges providing you with a [**Markov equivalence class**](https://www.youtube.com/watch?v=nnjKCtdORwY) of possible graphs (hence the name, once again).

In my experience, despite its theoretical foundations, GES often underperforms compared to other methods when applied to real-world data.



Image by [Antoni Shkraba](https://www.pexels.com/@shkrabaanthony/) @ [Pexels](https://www.pexels.com/photo/compass-in-a-case-6969337/)

**Functional methods**

In a sense, most functional methods can be considered score-based methods, because they involve some sort of goodness-of-fit computation at some point. On the other hand, their mechanics differ from the latter. Classic functional methods, like [**LiNGAM**](https://sites.google.com/view/sshimizu06/lingam) (**Linear Non-Gaussian Acyclic Model**; Shimizu et al., 2006) leverage the distributional asymmetries in the data rather than (greedy) edge search in order to retrieve the causal relations from the data.

**LiNGAM**

L**iNGAM** (Linear non-Gaussian Acyclic Models) has been first introduced by Shohei Shimizu and colleagues in 2006. The original method uses [**independent component analysis**](https://en.wikipedia.org/wiki/Independent_component_analysis) (**ICA**) to retrieve information about the data generating process. Its later variant — DirectLiNGAM (Shimizu et al., 2011) — leverages a linear model and a kernel-based independence measure.

Two **main assumptions** behind LiNGAM are:

* **No hidden confounding**
* **All (or all but one) error terms are non-Gaussian**

That said, various extensions have been proposed to LiNGAM, allowing to apply the model to scenarios with **hidden confounding** (Hoyer et al., 2008) or **cycles** (Lacerda et al., 2008).

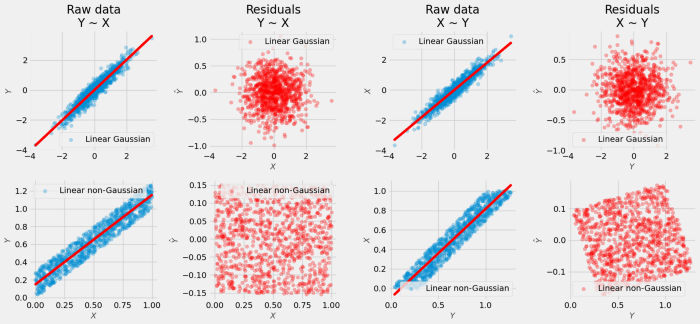
The main idea behind LiNGAM is **relatively simple**. Imagine a simple linear system with just two variables ***X*** and ***Y***, where ***X*** causes ***Y***. You can run linear regression on this data in two directions: regressing ***Y*** on ***X*** (***X → Y***) or regressing ***X*** on ***Y*** (***Y → X***). If error terms in your data are Gaussian these models will tell you nothing about the causal direction. The residuals for both models will be completely independent.

Yet, if your error terms are non-Gaussian…

We can break the symmetry!

It turns out that non-Gaussian data will force linear regression to return **correlated residuals** when we try to model the **non-causal direction**.

**Figure 2** presents the results of a simple experiment.



**Figure 2.** Raw data and residuals when regressing Gaussian and non-Gaussian data for the true model X -> Y. Left half: regressing Y on X; right half: regressing X on Y. Image by yours truly.

Notice that for Gaussian error terms (top row) residuals look very similar when we regress ***Y*** on ***X*** (left) and ***X*** on ***Y*** (right). For non-Gaussian data (bottom row) residuals are uncorrelated in the causal direction (***Y ~ X***; left), but become correlated in the non-causal direction (***X ~ Y***; right).

**Other methods**

O**ther methods** is a vast category! I picked one method for us to discuss today. The algorithm is called **GOLEM** and has been introduced by Ignavier Ng and colleagues in their NeurIPS 2020 paper (Ng et al., 2020). **GOLEM** can be classified as a **gradient-based method** (which means that it uses gradient descent for optimization) and in a sense it’s also a score-based method as we compute data likelihood scores on the way.

There are two variants of **GOLEM**:

* **GOLEM EV**
* **GOLEM NV**

Reisach et al. (2021) have shown that GOLEM EV outperforms its NV counterpart on unstandardized data.

**Shed no tears**

GOLEM is a successor of the **NOTEARS algorithm** (Zheng et al., 2018). NOTEARS was revolutionary because it was the first algorithm to frame **structure learning** as a purely **continuous optimization** problem (which solves the problem of the DAG search space explosion that grows superexponentially with the number of nodes).

Despite promising start, NOTEARS had been repeatedly shown to be unsuitable for stable causal discovery (Kaiser & Sipos, 2021; Reisach et al., 2021; Seng et al., 2022). Although GOLEM does not solve all the problems that NOTEARS comes with (*vide* Reisach et al., 2021) from my experience it works pretty well in certain cases in practice.

To learn more on **how GOLEM works** check [Ng et al. (2020)](https://arxiv.org/pdf/2006.10201.pdf).

**[Causal Python: 3 Simple Techniques to Jump-Start Your Causal Inference Journey Today](https://towardsdatascience.com/causal-kung-fu-in-python-3-basic-techniques-to-jump-start-your-causal-inference-journey-tonight-ae09181704f7" \t "_blank)**

**[Learn 3 techniques for causal effect identification and implement them in Python without losing months, weeks or days…](https://towardsdatascience.com/causal-kung-fu-in-python-3-basic-techniques-to-jump-start-your-causal-inference-journey-tonight-ae09181704f7" \t "_blank)**

[towardsdatascience.com](https://towardsdatascience.com/causal-kung-fu-in-python-3-basic-techniques-to-jump-start-your-causal-inference-journey-tonight-ae09181704f7" \t "_blank)

Ready to get your hands dirty?

**King of my castle**

Let’s start by introducing the hero of today’s blog post — **gCastle**.



**Figure 3.** gCastle logo. Source: <https://github.com/huawei-noah/trustworthyAI/tree/master/gcastle>

**g**[**Castle**](https://github.com/huawei-noah/trustworthyAI/tree/master/gcastle)is an open-source library developed by [Huawei Noah’s Ark Lab](https://www.noahlab.com.hk/#/home). The package provides us with an amazing and up-to-date toolkit for causal structure learning that includes:

* **Data-related tools** (incl. simulation and pre-processing)
* Extensive set of **causal discovery algorithms**
* **Evaluatioin metrics**

The full list of currently available algorithms is available [**here**](https://github.com/huawei-noah/trustworthyAI/tree/master/gcastle)**.**

Up to my best knowledge this is **the** **biggest**, **most complete** and **most up-to-date** list of causal discovery algorithms you can find in **any open-source causal Python package**.

And you know what’s best? This list is **growing systematically**!

A great advantage of **gCastle** is that it provides us with a unified, very intuitive and elegant API for interacting with various causal models. Forget about loading five different causal discovery packages, where two of them are porting to R and each has a completely different API in order to compare a couple of classic algorithms. **gCastle** makes this so much easier!

But don’t take my word for granted. See for yourself.

**Let’s do it!**

In this section we’ll implement and comapre four causal discovey algorithms using **gCastle**:

* **PC**
* **GES**
* **ICA-LiNGAM**
* **GOLEM**

Let’s start with imports and some basic settings.

**Code block 1.** Imports and basic settings

We import os module to modify gCastle’s environmental variable and set the the library’s backend to PyTorch. We import OrederedDict to organize our experiemnts nicely and networkx to visualize graphs.

Next, we have several objects from castle (this is how **gCastle** shows up in Python’s namespace):

* GraphDAG for plotting adjacency matrices
* MetricsDAG for automated metrics computation
* DAG and IIDSimulation for generating simulated data
* Models: PC, GES, ICALiNGAM and GOLEM

**Starting simple**

We’ll start by implementing the example from **Figure 2**. We’ll generate some linear Gaussian data according to the graph **A** in **Figure 2** and use the **PC algorithm** to recover the structure of this graph from the data. Our dataset will consist of 1000 samples.

**Code block 2.** Generating random 1000 samples according to the structure presented in Figure 2A.

Let’s instantiate and fit the model and print out the learned graph. As we said before, **gCastle** provides us with a unified training API for causal discovery models. In order to fit a model, we use the model’s .learn() method.

**Code block 3.** Instantiating and fitting the PC algorithm. After the model is trained, we print out the learned structure.

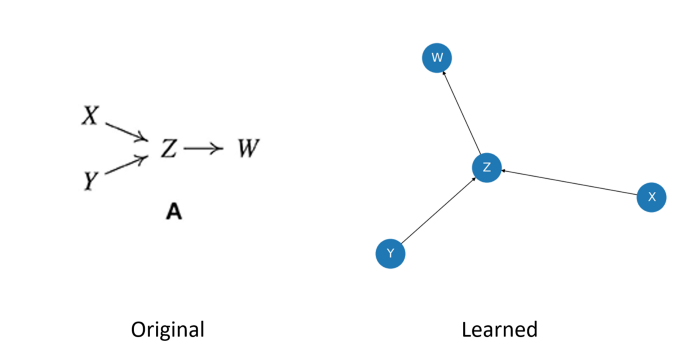
Note that the learned graph is represented as an [**adjacency matrix**](https://en.wikipedia.org/wiki/Adjacency_matrix).

Let’s plot the learned graph and compare it to the original.

**Code block 4.** Plotting the learned graph.

We use networkx to cast our adjacency matrix to a nx.DiGraph() object and plot it. On the way we relabel the nodes for ease of interpretability.

**Figure 4** presents the learned graph (right) and the ground truth (left).



**Figure 4.** Original graph from Figure 2A (left) and the graph learned by the PC algorithm (right). Sources: Glymour et al., 2019 (left), yours truly (right).

Both representations look different, but they represent the same graph (if in doubt, write down the list of directed edges for each of them; are the lists the same?).

This means that PC was able to retrieve the structure perfectly! Congrats PC! 🎉

**Stormy waters**

The PC algorithm worked really well in the first example. That’s great news! Now it’s time to see how it performs in a more complex situation.

We will explore the capabilities of the PC algorithm and see how it fares against other three algorithms.

Let’s start with generating a random DAG with 10 nodes and 15 edges. We’ll use a [scale-free network](https://en.wikipedia.org/wiki/Scale-free_network) to generate our graph. Then, we’ll produce three different datasets using this DAG as a structural model:

* **Linear Gaussian**
* **Linear exponential**
* **Non-linear quadratic**

and store them in a Python dictionary. See **Code block 5** for implementation.

**Code block 5.** Generating a random DAG and three distinct datasets.

Note that in the double for-loop we create a new instance of the IIDSimulation object for each set of conditions (linear-Gaussian, linear-exponential, etc). You can verify that our datasets belong to the class castle.datasets.simulator.IIDSimulation by checking the printout at the bottom of **Code block 5**.

We’re now ready to run our comparison. We begin by creating a Python dictionary with the names of the algorithms as keys and **gCastle** objects representing the algorithms as values.

Next, we loop through the datasets and train each model on each dataset. Note that we instantiate GOLEM differently compared to the other models in order to determine the number of iterations for the algorithm. Check the implementation in **Code block 6**.

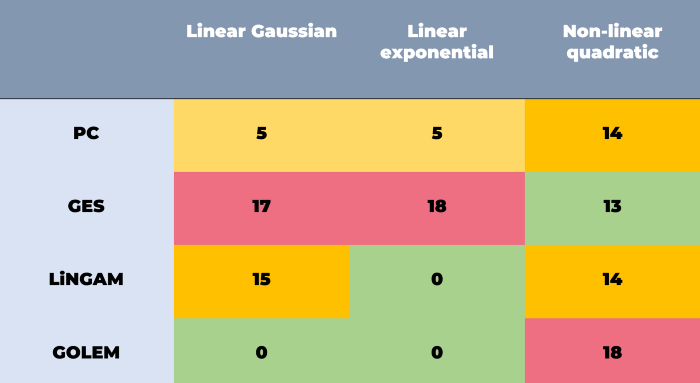
**Code block 6.** Training all four models on the three datasets and printing out the results.

In each iteration we plot the true DAG, the discovered DAG and print out six evaluation metrics:

* [**False discovery rate**](https://en.wikipedia.org/wiki/False_discovery_rate)(**FDR**)
* [**Recall**](https://en.wikipedia.org/wiki/Precision_and_recall)
* [**Precision**](https://en.wikipedia.org/wiki/Precision_and_recall)
* [**F1 score**](https://en.wikipedia.org/wiki/F-score)
* [**Structural Hamming distance**](https://rdrr.io/cran/pcalg/man/shd.html)(**SHD**)³
* **Number of undirected edges**

**Results**

**Figure 5** presents the results in terms of SHD. For complete results check the notebook (link below).



**Figure 5.** SHD for each dataset/model combination. Image by yours truly.

SHD of zero means that the model was able to recover the **true structure perfectly**. As we can see, GOLEM performs the best on average, but fails badly on non-linear quadratic dataset. This dataset is the most difficult for all algorithms. Note how LiNGAM performs good on linear exponential data, while failing on other two datasets. The reason for this is that the linear exponential dataset is the only one that meets the model’s assumptions (linear, non-Gaussian, acyclic). GES heavily underperforms comparing to other models, yet gives the best result for the most challenging dataset. That said, we need to remember that SHD does not tell the full story.

I encourage you to check the notebook for full results and analyze the data from other perspectives (e.g. false discovery rate or precision). Depending on your use case it might be the case that FDR is moere important to you than overall correctness.

**Wrapping it up**

Congratulations! You made it to the end! 👏🏼👏🏼👏🏼

Let’s make a quick recap!

In today’s blog post we learned about four families of **causal discovery methods**. We discussed some of their main advantages and disadvantages and implemented them in Python using the awesome **gCastle** library.

After reading this blog post and accompanying code you should be able to **apply** the discussed **techniques** to your **own datasets** and **problems**.

**Final thoughts**

Before we conclude I want to share something with you. Causal discovery is a **hard problem** and it’s good to always be **extra cautious** when using causal discovery methods. Be sure to check your results twice and use any available validation methods (e.g. [**refutation tests**](https://towardsdatascience.com/causal-kung-fu-in-python-3-basic-techniques-to-jump-start-your-causal-inference-journey-tonight-ae09181704f7)) before moving to the next stage and remember that in the **real world** it’s hard to get **any** **guarantees** from causal discovery methods, especially if you cannot determine if all relevant variables are present in your dataset.

If you have an opportunity to perform even a **minimal intervention** on your system of interest, data from such an intervention can be used to validate your causal graph in a more reliable fashion. There are interesting methods that allow you to pick the optimal interventions in such scenarios, but that’s a story for another post.