POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in INGEGNERIA MATEMATICA

Generalization of the PC algorithm for non-linear and non-Gaussian data and its application to biological data

Relatore: Prof. Anna Maria Paganoni

Co - Relatori: Prof. Lorenz Wernisch
Dott. Francesca Ieva

Tesi di Laurea Magistrale di:

Nina Ines Bertille Desgranges
matr. 799924

Anno Accademico 2013-2014

Abstract

This thesis arises in the context of finding causal relationships among observed data. It will mostly focus on data coming from biological measurement of protein or gene expression. The PC is a widely used algorithm for learning graphical models. The causal structures are created evaluating the dependency or conditional dependency among the variables. Its output is either a DAG or a class of DAGs. It has been developed for linear data with Gaussian noise, which makes it less suitable when applied to biological data, which are often far from having these features. In this thesis we develop methods (i) to solve the problem of linearity/Gaussianity and (ii) to restrict the possible outputs of the algorithm. In order to overcome the problem (i), we use conditional dependency tests which characterize probabilistic independence: we use quantities which are zero if and only if the data are independent and not only uncorrelated. Two families of test are investigated: a non-parametric kernel based test of conditional independence (kPC) and a test based on the Brownian distance correlation (bPC). In the kPC, thanks to the kernel, the data are implicitly embedded in a space where the data are more linear. In the bPC the Brownian correlation measures the weighted distance between $f_{X,Y}$ and $f_X f_Y$. In order to solve problem (ii), we describe the data with a weakly additive noise model and we use the residuals of the non-linear regression to find more causal directions than with the PC algorithm. The developed algorithms are tested on data on the data used in Sachs et al. 2005.

Sommario

Questa tesi si colloca nell'ambito di sviluppare algoritmi per la ricerca di relazioni di causalità tra dati osservabili. Più precisamente, saranno presi in considerazione dati quali espressione proteica o genomica. L'algoritmo PC è un metodo tra i più usati per ricavare relazioni di dipendenza e rappresentarle tramite grafi. Le connessioni tra i nodi presenti nei grafi sorgono valutando relazioni di dipendenza o dipendenza condizionata tra variabili. L'output dell' algoritmo può essere sia una classe di DAG, che un certo DAG. Quest' algoritmo è stato sviluppato per dati che hanno come caratteristica l'essere lineari e avere rumore gaussiano; queste proprietà sono però difficili da trovare in dati provenienti dalla biologia. In questa tesi sviluppiamo metodi (i) per risolvere il problema della linearità/gaussianità e (ii) per restringere il possibile output di PC. Per risolvere il problema (i) usiamo test che caratterizzano l'indipendenza probabilistica in tutti i sensi: cerchiamo quantità che siano zero se e solo se le variabili sono indipendenti. Vagliamo due diverse famiglie di test: un test non parametrico basato sui kernel (kPC) ed un altro che considera la distanza browniana (bPC). In kPC, grazie al kernel, i dati vengono analizzati in un nuovo spazio, nel quale le relazioni sono più lineari ed è quindi più semplice trovare dipendenze. In bPC, la correlazione Browniana misura la distanza pesata tra $f_{X,Y}$ e $f_X f_Y$. Per risolvere il problema (ii), i dati vengono descritti con un weakly additive noise model e i residui di una regressione (non-lineare) sono usati per trovare il maggior numero possibile di direzioni di causalità. Infine, per essere confrontati, gli algoritmi sviluppati sono testati su dati usati in Sachs et al. 2005

Contents

1 INTRODUCTION

Causality has always played an important role in our everyday life. Most of the time, we do not even think about the underlying reasons that make us behave in a specific way. When, for example, before going out we put on our coat, many causal relationships are involved. We know it is cold outside, which leads us to take our coat; once outdoors, wearing the coat will cause us to feel warm. Many mathematicians, statisticians and philosophers have for a long time been wondering about this topic. Among them is Sewall Weight, who has always been interested in causality and genetics, and developed path analysis (Wright 1918; Wright 1921; Wright 1934) in the 1920s. The ambition of many scientists has been to find a link between the data which could be collected, and the causal relationships that ought to be found. We can affirm that their goal can be achieved under certain conditions. As Judea Pearl said in (Pearl 2000) *"...causality has been mathematized"*. Moreover, many researchers as Steffen Lauritzen, Clark Glymour, Peter Spirtes (*Probabilistic reasoning in intelligent systems: networks of plausible inference*; Spirtes, Glymour, and Scheines 2000; Lauritzen 1996) are occupied in developing new methods or expanding existing ones in oder to enlarge our knowledge. In my small way, that is what I am about to do with this thesis. The scheme developed by the pioneer of this doctrine when dealing with causality is the following:

I. Collect the observations.

II. Build a causal knowledge.

III. Infer about the consequences of manipulating variables.

In this project, I will mainly investigate the first two points. In order to analyze causal relationships, we will adopt graphs. Graphs facilitate the representation of causal links and the description of joint probability distributions. Along with the development of this theory, discovering causality among phenomena has become increasingly important in many fields. Just to mention some, we have the biological sciences, epidemiology, artificial intelligence, social and behavioral sciences. In this project I will handle data

coming from biology. Biological data which can be measured, are data on gene expression, gene activity and protein expression. Each of them is regarded as a variable. The aim of studying networks in biology is to find any kind of connection among variables. The detection of causal links among those kinds of data can be extremely useful. Thanks to some advanced techniques it is possible to measure the state of a variable. Applying those methods to detect causality, we can elucidate what the cause - effect relationships are. Starting from this, many conjectures about the healthiness of a cell, a patient, a illness arise, and help medical research.

To go more in detail, this work will focus on developing some algorithms to learn graphical models with non-Gaussian and non-linear data. We will use a "PC-style" search combined with tests which characterize probabilistic independence. To restrict the possible class of output, we will assume the data generated by a weakly additive noise model. In order to implement tests to discover whether the data are independent or not we rely on two quantities:

1. Hilbert Schmidt independence criterion (HSIC)
2. Brownian distance

An algorithm which uses the HSIC has already been implemented in Tillman 2009, we will name it "kPC Permutation - Cluster". In this thesis we develop two more ways to use the HSIC in independent test: "kPC Permutation - Residuals" and "kPC Residuals". Furthermore, we investigate how the free parameters, which have to be fixed when calculating HSIC, affect the output of the test. We also develop the brand-new bPC, which independent test relies on the Brownian distance. In order to compare the algorithms and evaluate their performance we use in a first moment simulated data and then real data of protein expression. The data are the one used in Sachs et al. 2005.

The structure of the work is the following:

- in Chapter 2 the PC algorithm is described and the kPCs and bPC are developed and analyzed with simulated data.
- in Chapter 3 the real data are illustrated and the PC, kPCs and bPC are tested on these data.
- in Chapter 4 the conclusions are drawn and some ideas about possible extension are illustrated.

All the developed algorithms have been implemented using the statistical software R (R Core Team 2014).

2 ALGORITHMS

2.1 Probabilistic graphical models

A graph $\mathbf{G}$ is an ordered pair $\langle \mathbf{V}, \mathbf{E} \rangle$ where $\mathbf{V}$ is a set of vertices (or nodes) and $\mathbf{E}$ is a set of edges (or arcs) (Spirtes, Glymour, and Scheines 2000). Two vertices are adjacent if they are connected by an edge. An edge can be either directed or undirected .

Figure 2.1: Directed edge.

Figure 2.2: Undirected edge.

In the case of directed edges (Figure 2.1), we say that V_1 is a parent of V_2 and V_2 is a child of V_1. In general, given a set of vertices $\mathbf{V}$, we represent the set of the children of $\mathbf{V}$ in the graph $\mathbf{G}$ as $Ch_{\mathbf{G}}(\mathbf{V})$, and the set of its parents as $Pa_{\mathbf{G}}(\mathbf{V})$. The number of edges entering in the node V is called indegree, the number of edges leaving V is the outdegree. The degree of an edge V is the total number of edges adjacent to V. Considering a graph $\mathbf{G}$ and a sequence of nodes $\{V_1, \ldots, V_n\} \subset \mathbf{V}$, we say that in a directed path all the edges are directed and are pointing from V_i to V_{i+1} for $i \in \{1, \ldots, n-1\}$. In an undirected path the nodes V_i, V_{i+1} are adjacent for $i \in \{1, \ldots, n-1\}$. A path is said to be acyclic if it does not contains a vertex more than once, otherwise it is cyclic. An ancestor of a vertex V is any vertex V_1 such as there is a directed path from V_1 to V; a descendant of a vertex V is any vertex V_2 such as there is a directed path from V to V_2. We say that a graph is directed if all its edges are directed, it is mixed if it has directed and undirected edges and it is undirected if it has only undirected edges. A graph is complete if all the nodes are linked to each other. A collider (V-structure) is a triple $\langle V_1, V_2, V_3 \rangle \subset \mathbf{V}$ such as $\{V_1, V_3\} \in Pa_{\mathbf{G}}(V_2)$. The collider is said to be immoral (unshielded collider) if there is no edge between V_1 and V_3 . A chain is a triple $\langle V_1, V_2, V_3 \rangle \subset \mathbf{V}$ such that $V_1 \to V_2$ and $V_2 \to V_3$. A fork is a triple $\langle V_1, V_2, V_3 \rangle \subset \mathbf{V}$ such that $V_2 \to V_1$ and $V_2 \to V_3$ as illustrated in Figures 2.3-2.5.

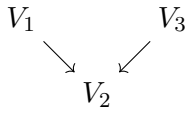


Figure 2.3: Unshielded collider.

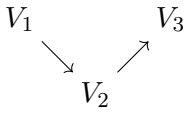


Figure 2.4: Chain.

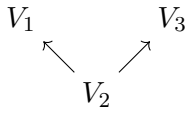


Figure 2.5: Fork.

In the next pages we will mostly focus on directed acyclic graphs (DAG), which are directed graphs that contain no cyclic paths, and partially directed acyclic graphs (PDAG), which are DAGs that contain undirected edges.
A DAG can be used to represent causal relationships between nodes. What we need to investigate whether or not there is this relationship, is how to link the concept of dependence and causality.
We start analyzing Figure 2.6 and assuming that rain and watering the garden are the only causes of the mud in the garden. As we know, and as we can see in the picture, both rain and watering the garden cause mud, but they are not related. Furthermore, knowing that it has not rained, if we know that there is mud in our garden, we can infer about the garden watering. That means that in some way rain and garden watering are linked when we know something about mud.
A concept which is fundamental for our purpose is the d-separability (directed separation).

Definition 1. *Given a causal graph* $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ *if* V_1 *and* $V_2 \in \mathbf{V}$ *and* $\mathbf{Q}$ *is a set of vertices in* $\mathbf{G}$ *that does not contain* V_1 *and* V_2 *then* V_1 *and* V_2 *are d-separated given* $\mathbf{Q}$ *in* $\mathbf{G}$ *if and only if there exist no undirected path* U *between* V_1 *and* V_2 *such that*

i. every collider on U *is either in* $\mathbf{Q}$ *or else has a descendant in* $\mathbf{Q}$.

ii. no other vertex on U *is in* $\mathbf{Q}$.

To explain this concept with simpler words, here is the procedure to follow if we want to know whether V_1 and V_2 are d-separate given $\mathbf{Q} \in \mathbf{V} \backslash \{V_1, V_2\}$

1. Find all the undirected paths from V_1 to V_2.
2. Check one by one all paths
   - If not all the colliders or the descendants of colliders which are in the path are also in $\mathbf{Q}$ the path is blocked.

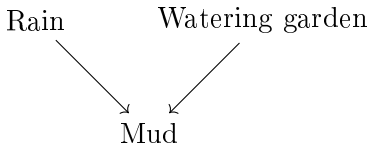


Figure 2.6: Example causality.

- If at least one non-collider which is in the path is also in $\mathbf{Q}$ the path is blocked.

3. If all the paths are blocked V_1 to V_2 are d-separated given $\mathbf{Q}$.

According to that definition, vertices can have the property to be ON or OFF in a certain path. If they let the information pass, they are ON (the middle node in forks, and chains), otherwise they are OFF (the middle node in colliders). Conditioning on a set of nodes, makes the status of those nodes on which we are conditioning change. This explains why, when we know that there is the mud, we can know whether someone has watered the garden or not: the collider blocks the information between rain and garden watering but, conditioning on it, the information can flow. For example, looking at

$$\begin{array}{ccccccc} X & \rightarrow & Y & \rightarrow & Z & \rightarrow & W \\ & & & & \uparrow & & \\ & & & & V & & \end{array}$$

Figure 2.7: Example d-separation.

Figure 2.7 we want to infer whether:

i. X is d-separated from V given Z: the only path between X and V is $X \rightarrow Y \rightarrow Z \leftarrow V$. The vertex Y is ON and the vertex Z is OFF. Conditioning on Z, we change its status so the flow of information passes. The two nodes are not d-separated.

ii. X is d-separated from W given Z: the only path between X and W is $X \rightarrow Y \rightarrow Z \rightarrow W$. Both vertices Y and Z are ON. Conditioning on Z, we change its status so the flow of information is blocked. The two nodes are d-separated.

In a graphical model, the nodes of a graph are variables which are characterized by a probability distribution. The structure of some graphs, such as DAG, entails the conditional independent relationship among the variables. Assuming that X and Y are two variables, f_X and f_Y their density and $f_{X,Y}$ their joint density, X and Y are independent if their joint density is equal to the product between the density of X and the density of Y:

$$f_{X,Y} = f_X f_Y.$$

Generalizing this concept, X and Y are conditionally independent given a set of variables $\mathbf{Z}$ if the joint density of X and Y given $\mathbf{Z}$ is equal to the product between the density of X given $\mathbf{Z}$ and the density of Y given $\mathbf{Z}$:

$$f_{X,Y|\mathbf{Z}} = f_{X|\mathbf{Z}} f_{Y|\mathbf{Z}}.$$

For a graph $\mathbf{G}$ we then have its nodes $\mathbf{V}$ and the probability distribution over the nodes $P(\mathbf{V})$, and we will denote it as: $\langle \mathbf{G}, P \rangle$. A DAG $\mathbf{G}$ and a probability distribution $P(\mathbf{V})$ over the node $\mathbf{V}$ of $\mathbf{G}$ satisfy the Causal Markov condition if and only if

$$\forall X \in \mathbf{V}, X \perp\!\!\!\perp [\mathbf{V} \setminus (Desc(X) \cup Pa_{\mathbf{G}}(X))] \mid Pa_{\mathbf{G}}(X).$$

We can write the joint density function satisfying the Markov condition over the nodes $\mathbf{V}$ as:

$$f(\mathbf{V}) = \prod_{X \in \mathbf{V}} f(X \mid Pa_{\mathbf{G}}(X)).$$

The Markov condition states that to learn about an event we only need to know its direct causes; knowing more about the undirect causes does not add any more information. The graphs are Markov equivalent when they imply the same independence relations defined by the Causal Markov property. Markov equivalent graphs have the same skeleton and consist of the same set of immoralities (Flesch and Lucas 2007). Two other interesting properties of DAGs and $P(\mathbf{V})$ are the Causal Minimality condition and the Faithfulness. We say that $\langle \mathbf{G}, P \rangle$ satisfy the Minimality condition if and only if for all proper subgraphs $\mathbf{H}$ of $\mathbf{G}$ with vertices $\mathbf{V}$, $\langle \mathbf{H}, P \rangle$ do not satisfy the Causal Markov condition. A probability distribution $P(\mathbf{V})$ over a DAG $\mathbf{G}$ is Faithful if and only if all the conditional independence relations entailed by P arise from the Causal Markov condition applied to $\mathbf{G}$. It might

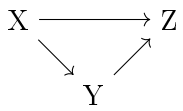


Figure 2.8: Example Faithfulness.

happen that the Causal Markov property entails other types of conditional independence, which could "erase" the dependence between variables that are dependent according to the Markov property. This might be the case when the quantitative causal effects between two variables are exactly the opposite in two different paths. Considering for example Figure 2.8: X and Z are not d-separate given Y. If we calculate the Pearson partial correlation $\rho_{XZ|Y} = \dfrac{\rho_{XZ} - \rho_{XY}\rho_{YZ}}{\sqrt{1-\rho_{XY}^2}\sqrt{1-\rho_{YZ}^2}}$, it turns out that it is equal to zero when $\rho_{XZ} = \rho_{XY}\rho_{ZY}$. If this happens, the graph is unfaithful.
We can now find a link between the causal dependency (d-separation) and the statistical dependency ($f_{X,Y} = f_X f_Y$). This is the key point that enables us to use graphical models to represent the conditional independent relationships among variables. The main result is:

Theorem 1. *P(***V***) is faithful to DAG* **G** *with vertex set* **V** *if and only if for all disjoint set of vertices* **X**, **Y** *and* **Z**, **X** *and* **Y** *are independent conditional on* **Z** *if and only if* **X** *and* **Y** *are d-separated given* **Z**

2.2 PC ALGORITHM

In the literature there exist several algorithms used for learning graphical models. They can work backwards: they start with a complete graph, and they remove edges according to some rules. There exist also forward algorithms: the starting point is a graph with no edges, and they keep going on adding edges until all the necessary ones have been added. Moreover, we can choose among different criteria to check which graph best represents the conditional independence among the variables. We can use score based methods: they consider a score function and select the graph which minimizes or maximizes that score. Typical score function are the BIC, AIC or the posterior probability given the data(Pe'er et al. 2001). Another possibility is a constraint-based method: the absence or presence of edges is entailed by the conditional independence among variables.
The PC (Spirtes, Glymour, and Scheines 2000) is an algorithm widely used to discover causal structures. It is a constraint based method which uses a structured backward search. Given input data, the output is either a class of DAGs, described as a PDAG, or a DAG. In order to have as output a graph or a class of graphs faithful to the population distribution, the following assumptions must be fulfilled:

- The set of observed variables **V** is causally sufficient: every common cause of any two or more variables either is in **V** or has the same value for all units in the population.
- Every unit in the population has the same causal relation among variables.
- The distribution of the observed variables either is faithful to an acyclic directed graph of the causal structure or linearly faithful to such a graph.
- The statistical decisions required by the algorithm are correct for the population.

The PC algorithm consists of 3 phases which are briefly described in Algorithm 1 - 2 - 3

Phase 1. Skeleton

Phase 2. Collider

Phase 3. Transitive

Algorithm 1 SKELETON phase

Input: data, α

Output: G_{und}: skeleton of the DAG & Sepset

1: Form the complete undirected graph G_{und}
2: $d = 0$
3: **repeat**
4: **for** each ordered pair of adjacent vertices X and Y in G **do**
5: **if** $|adj(X, G_{und})\backslash Y| \geq d$ **then**
6: **for** each subset $\mathbf{S} \subseteq adj(X, G_{und})\backslash Y$ and $|\mathbf{S}| = d$ **do**
7: test $X \perp\!\!\!\perp Y \mid \mathbf{S}$ and calculate the p-value p of test 2.1
8: **if** $p \geq \alpha$ **then**
9: remove the edge $X - Y$ from G_{und}
10: record $\mathbf{S}$ in Sepset(X,Y)
11: **break** the for loop at line 6
12: **end if**
13: **end for**
14: **end if**
15: **end for**
16: $d = d + 1$
17: **until** $|adj(X, G_{und})\backslash Y| < d$

As we can see in the skeleton phase (Algorithm 1), the input are the data and a threshold α. G_{und} indicates an undirected graph while $Adj(X, G_{und})$ indicates all the nodes adjacent to X in G_{und}. In this phase it is tested whether each pair of variables is linked by an edge or not. The aim of the test is to establish if two variables X and Y are independent or conditionally independent given a subset of the remaining variables $\mathbf{S}$. The test is the following:

$$H_0 : X \perp\!\!\!\perp Y \,|\, \mathbf{S} \text{ against } H_1 : X \not\perp\!\!\!\perp Y \,|\, \mathbf{S} \tag{2.1}$$

$\mathbf{S}$ may be the null set. The key points of this phase are that:

1. The algorithm does not perform the conditional interdependency test given all the possible subsets. The tests that are performed are only the ones where the conditioning set is a subset of the adjacent nodes of one of the two tested variables.

2. The cardinality of the conditioning set increases step by step.

Furthermore, the algorithm keeps in memory the subset which causes the separation of the two variables (called Sepset). Avoiding to test all the conditioning subsets is computationally very efficient. The output of this phase is an undirected graph G_{und} and Sepset. In Phase 2 some edges are orientated. The input is the undirected graph G_{und} and Sepset. This step is a direct consequence that the unshielded colliders are the only triples in which

Algorithm 2 COLLIDER phase

Input: G_{und} and Sepset
Output: G_{coll}

1: **for** each triple in G_{und} such as X and Y are adjacent, Y and Z are adjacent, X and Z are not adjacent **do**
2: **if** $Y \notin \text{Sepset}(X, Y)$ **then**
3: orient the triple $X - Y - Z$ in this way $X \rightarrow Y \leftarrow Z$ in the graph G
4: **end if**
5: **end for**

the nodes at the two ends are independent, but dependent conditionally on the middle one. The output is a mixed graph G_{coll}. Finally, in the last step,

Algorithm 3 TRANSITIVE phase

Input: G_{coll}
Output: G

1: **repeat**
2: **if** $X \rightarrow Y$ and $Y - Z$ and X and Z are no adjacent **then**
3: orient the triple in this way $X \rightarrow Y \rightarrow Z$ in G
4: **end if**
5: **if** $X - Y$ and there is a directed path from X to Y **then**
6: orient $X \rightarrow Y$ in G
7: **end if**
8: **until** until no more edges can be oriented

some others edges are orientated thanks to the transitive property. The input is the graph G_{coll} and the output is the final graph with more edges oriented.
The main trend in augmenting α is to find more edges, because the threshold to refuse independence becomes higher. As the variables are affected with increasing noise, the PC algorithm has more difficulty in finding the underlying dependencies. When a graph is characterized by different degree of dependency, for example when in linear relationship the coefficients have different orders of magnitude, it is possible that the algorithm is not able to find all dependencies.
In order to analyze biological data, which are highly non-linear and may not have Gaussian noise, this algorithm is not appropriate (Voortman and Druzdel 2008). This is due to the fact that the test which is performed, the `gaussCItest` in `R` (Kalisch 2012; Hauser 2012) is suitable for linear data with Gaussian noise. The test consists of the following phases:

1. Compute the partial correlation r of the samples X and Y given $\mathbf{S}$.

2. Calculate the z-Fisher transformation as

$$Z = \frac{1}{2}\sqrt{n - |\mathbf{S}| - 3}\, \log\left(\frac{1+r}{1-r}\right)$$

where n is the number of observations and $|S|$ is the number of variables in the conditioning set.

3. Knowing that under the hypothesis $X \perp\!\!\!\perp Y \mid \mathbf{S}$, $Z \sim \mathcal{N}(0,1)$, calculate the corresponding p-value of the test.

Another issue which arises is the erasing of shielded collider. In real data, unfortunately, not all the hypothesis which are necessary are fulfilled. This can lead to have an edge that at the end of the Phase 2 is directed as $\leftrightarrow$ due to the presence of colliders. The problem is that in the following step this arrow is seen as an undirected arrow and may be redirected as $\rightarrow$ or as $\leftarrow$. Finally the PC algorithm is not able to orient edges when some structures as the one in Figure 2.9 are found. This is due to the fact that, whatever the

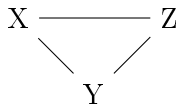


Figure 2.9: Example Triangle.

orientation of the arrows, the algorithm cannot differentiate among distributions of variables with edges orientated in one way or the other way around. In the next two sections I will investigate how to solve the problem of non-linearity and non Gaussianity in biological data. We will consider a dependency test which takes into account non-linearity and non-Gaussianity. Section 2.6 will analyze a possible way to discover more causal relationships.

2.3 Kernel PC

The first approach to overcome the problems with the PC is to use a kernel-based non-parametric test for independence which can be generalized to a conditional version. We analyze this method, not only because it is able to recover non-linear features, but also because it will let us find an estimator which characterizes independence.

2.3.1 Kernel

Definitions and theorems for this sections are taken from (Shawe-Taylor and Cristianini 2004; Tillman 2009; Gretton et al. 2005).
A kernel is a function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ such as $\forall\ x, z \in \mathcal{X}$ satisfies

$$k(x,z) = \langle \phi(x), \phi(z) \rangle_{\mathcal{H}}$$

where $\mathcal{X}$ is the space of all the random variables, $\phi : \mathcal{X} \to \mathcal{H}$ is a mapping from $\mathcal{X}$ to a Hilbert feature space $\mathcal{H}$. I will refer to this function as embedding function, feature map or embedding map. The kernel measures the distance between data after some transformations on the data have been applied. The aim of using kernel functions is that, thanks to the feature map, the data are embedded in a space where linear patterns can be found. Closely related to

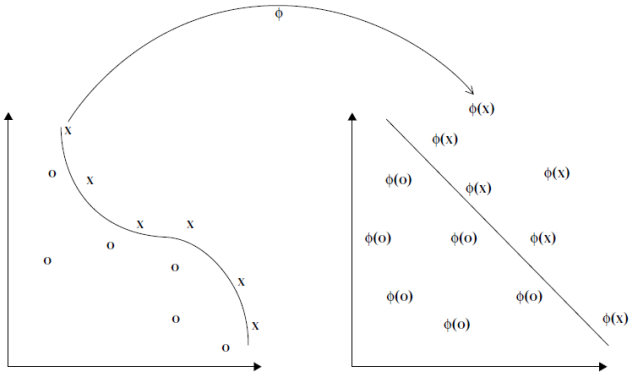


Figure 2.10: Example of the embedding function ϕ.

the kernel function is the associated Gram matrix K, also known as kernel matrix, with entries $K_{i,j} = k(x_i, x_j)$. A symmetric matrix is positive semi-definite if its eigenvalues are all non-negative. Gram matrices are always positive semi-definite. A function satisfies the finitely positive semi-definite property if it is symmetric and the matrices formed by restriction to any finite subset of the space $\mathcal{X}$ are positive semi-definite. That means that $\forall N \in \mathbb{N},\ \ \forall\{x_1, \ldots, x_N\} \in \mathcal{X},\ \ \forall\{\alpha_1, \ldots, \alpha_N\} \in \mathbb{R}^N$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) = \boldsymbol{\alpha}^T K \boldsymbol{\alpha} \geq 0.$$

Thanks to the Characterization theorem, we can affirm that this property characterizes kernel.

Theorem 2. Characterization Theorem *A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is either continuous or has a finite domain, can be decomposed into a feature map ϕ into an Hilbert space $\mathcal{H}$ applied to both its arguments followed by the evaluation of the inner product in $\mathcal{H}$ if and only if it satisfies the finitely positive semi-definite property.*

Furthermore, given a function k which satisfies the finitely semi-definite positive property, k satisfies the following reproducing property:

$$\phi(x) = \langle \phi(\cdot), k(x, \cdot) \rangle_{\mathcal{H}} \quad \forall \phi \in \mathcal{H}_k, \quad \forall x \in \mathcal{X}.$$

We will refer to its corresponding feature space $\mathcal{H}_k$ as its Reproducing Kernel Hilbert Space (RKHS). It is important to point out that, assuming to have a function which satisfies the finitely semi-positive property, we can deal with data in an embedded space that we do not need to explicitly know. We can treat $k(x, \cdot)$ as a feature map and, according to the reproducing property:

$$k(x, z) = \langle k(x, \cdot), k(z, \cdot) \rangle_{\mathcal{H}_k} \quad \forall x, z \in \mathcal{X}.$$

Thanks to the Moore Aronszajn theorem, we can show that all kernels which are symmetric and positive definite are reproducing kernel and uniquely define corresponding RKHS. In the next pages we will focus our attention on symmetric positive definite kernels.
Given two separable RKHS spaces $\mathcal{H}$ and $\mathcal{G}$ and their orthonormal bases u_i and v_j we consider the linear operator $C : \mathcal{G} \to \mathcal{H}$. Provided the sum converges, the Hilbert-Schmidt norm of C is:

$$\| C \|^2_{HS} = \sum_{i,j} \langle C v_j, u_i \rangle^2_{\mathcal{H}}.$$

A linear operator $C : \mathcal{G} \to \mathcal{H}$ is called Hilbert-Schmidt operator if its HS norm exists and is denoted as $HS(\mathcal{G}, \mathcal{H})$. Assuming $h \in \mathcal{H}$ and $g \in \mathcal{G}$, the tensor product $h \otimes g$ is defined as

$$(h \otimes g) f = h \langle g, f \rangle_{\mathcal{G}} \quad \forall f \in \mathcal{G}.$$

We will first define the independence test and then generalize it to a conditional independence test.

2.3.2 Test of Independence for kPC

Assuming that we have two random variables $\mathrm{X} \in \mathcal{X}$ and $\mathrm{Y} \in \mathcal{Y}$, let $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $l(\cdot, \cdot) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be symmetric reproducible kernels for $\mathcal{H}_\mathcal{X}$ and $\mathcal{H}_\mathcal{Y}$ (Gretton et al. 2005; Gretton et al. 2008). Assuming we have n samples, we can define the mean map, its empirical estimator and the cross covariance operator $C_{X,Y} : \mathcal{H}_\mathcal{Y} \to \mathcal{H}_\mathcal{X}$ as follow:

$$\mu_X = \mathbb{E}[k(x, \cdot)] \in \mathcal{H}_\mathcal{X} \qquad \hat{\mu}_X = \frac{1}{n} \sum_{i=1}^{n} k(x_i, \cdot)$$

$$\mu_Y = \mathbb{E}[l(y, \cdot)] \in \mathcal{H}_\mathcal{Y} \qquad \hat{\mu}_Y = \frac{1}{n} \sum_{i=1}^{n} l(y_i, \cdot)$$

$$C_{X,Y} = ([k(x, \cdot) - \mu_X] \otimes [l(y, \cdot) - \mu_Y]) \in HS(\mathcal{H}_\mathcal{Y}, \mathcal{H}_\mathcal{X})$$

The HSIC is the Hilbert Schmidt squared norm of the cross covariance:

$$\mathbb{H}_{X,Y} = \| C_{X,Y} \|^2_{HS} .$$

The cross-covariance operator can be seen as a generalization of the cross covariance matrix when we deal with an infinite dimensional feature spaces(Gretton 2014). To define the empirical estimate, assuming we have n samples, let K and L be the Gram matrices of $k(\cdot,\cdot)$ and $l(\cdot,\cdot)$. We are interested in the centered Gram matrices $\tilde{K}$ and $\tilde{L}$, thus we define $H = I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T$ where I_n is the identity matrix with n row and column and $\mathbf{1}_n$ is a vector full of ones of length n. So $\tilde{K} = HKH$, $\tilde{L} = HLH$. The empirical estimate of $\mathbb{H}_{X,Y}$ is:

$$\hat{\mathbb{H}}_{X,Y} = \frac{1}{n^2} tr(\tilde{K}\tilde{L}).$$

$\mathbb{H}_{X,Y} = 0$ if and only if X$\perp\!\!\!\perp$Y. It is very difficult to reach exactly the value 0 when we are testing two variables, hence we need to test whether the variables are independent or not. In the paper Gretton et al. 2008 are proposed two different ways to set the test $H_0 : x \perp\!\!\!\perp y$ against $H_1 : x \not\perp\!\!\!\perp y$

1. Permutation test
2. Gamma test

The desired output is a p-value to decide if the variables are independent or not.

Permutation Test

This approach use a Monte Carlo resampling technique. It consists on the following steps:

1. Calculate $\mathbb{H}_{X,Y}$.
2. Permute p times the ordering of Y obtaining $\{Y_{(1)}, \ldots, Y_{(p)}\}$ and then calculate the p different values of $\mathbb{H}_{X,Y_{(i)}}$, $i \in \{1, \ldots, p\}$.
3. Determine under which p-value α the variables are independent calculating the mean value of how many times $\mathbb{H}_{X,Y} \geq \mathbb{H}_{X,Y_{(i)}}$.

This method works because we assume that permuting Y destroys, if there is, the dependence between X and Y. If X and Y are independent $\mathbb{H}_{X,Y}$ would take values close to $\mathbb{H}_{X,Y_{(i)}}$. This leads to a high α. On the other hand, if X and Y are dependent, we would find a gap between $\mathbb{H}_{X,Y}$ and $\mathbb{H}_{X,Y_{(i)}}$ which lets us find a low p-value.

Gamma Test

In this approach we approximate the value of the asymptotic distribution of the empirical estimate HSIC(X, Y) under the null hypothesis as a two-parameters Gamma distribution. We then suppose that

$$n\text{HSIC} \sim \mathcal{G}\text{amma}(\alpha, \theta) \quad \Rightarrow \quad \text{HSIC} \sim \mathcal{G}\text{amma}(\alpha, \tfrac{\theta}{n})$$

where α is the shape parameter and θ is the scale parameter.

$$\alpha = \frac{\mathbb{E}[HSIC]^2}{Var(HSIC)}, \qquad \theta = \frac{nVar(HSIC)}{\mathbb{E}[HSIC]}$$

To compute this distribution we use empirical estimations of the mean and the variance under the null hypothesis. To find the p-value, we check to which quantile $\mathbb{H}_{X,Y}$ corresponds.

2.3.3 Test of Conditional Independence for kPC

Let $X, Y, \mathcal{X}, \mathcal{Y}, k(\cdot,\cdot), l(\cdot,\cdot), \mathcal{H}_X, \mathcal{H}_Y, \tilde{K}, \tilde{L}$ be as defined in 2.3.2. Suppose we want to test $X \perp\!\!\!\perp Y \mid \mathbf{Z}$ where $\mathbf{Z}$ is a set of variables defined in the space of the random variables $\mathcal{Z}$. Let $m(\cdot,\cdot) : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}$ be the symmetric reproducible kernel for $\mathcal{H}_{\mathbf{Z}}$ and $\tilde{M}$ be the centered Gram matrix for m. We define the extended variables $\ddot{X} = (X, \mathbf{Z})$ and $\ddot{Y} = (Y, \mathbf{Z})$. The conditional cross covariance is defined as:

$$C_{X,Y|\mathbf{Z}} = C_{\ddot{X}\mathbf{Z}} C_{\mathbf{Z}\mathbf{Z}}^{-1} C_{\mathbf{Z}\ddot{Y}}.$$

Therefore, we define a conditional version of the HSIC as

$$\mathbb{H}_{X \perp\!\!\!\perp Y|\mathbf{Z}} = \| C_{X,Y|\mathbf{Z}} \|_{HS}^2 .$$

X and Y are conditionally dependent given $\mathbf{Z}$ if and only if $\mathbb{H}_{X \perp\!\!\!\perp Y|\mathbf{Z}} = 0$ (Fukumizu et al. 2007). We can define an empirical estimator of $\mathbb{H}_{X \perp\!\!\!\perp Y|\mathbf{Z}}$ as

$$\hat{\mathbb{H}}_{X \perp\!\!\!\perp Y|\mathbf{Z}} = \frac{1}{2} tr(\tilde{K}\tilde{L} - 2\tilde{K}\tilde{M}(\tilde{M}+\epsilon I_n)^{-2}\tilde{M}\tilde{L} + \tilde{K}\tilde{M}(\tilde{M}+\epsilon I_n)^{-2}\tilde{M}\tilde{L}\tilde{M}(\tilde{M}+\epsilon I_n)^{-2}\tilde{M})$$

where ϵ is a regularization parameter. The calculation of $\hat{\mathbb{H}}_{X \perp\!\!\!\perp Y|\mathbf{Z}}$ is computationally very expensive. In Tillman, Gretton, and Spirtes 2009 some simplifications are introduced, assuming we have n samples:

1. Use an incomplete Cholesky decomposition for the kernel matrices K, L, M.

   $K = P_X G_X G_X^T P_X^T$ where $K \in n \times n, \quad P_X \in n \times n, \quad G_X \in n \times h$

   In this way $\tilde{K} = HKH = HP_X G_X G_X^T P_X^T H$ (repeat the same procedure with L and M)

   We have to fix the parameter h which we want to be much less than n.

2. Use a thin single value decomposition for $HP_X G_X$

   $HP_X G_X = U_X S_X V_X$ where $U_X \in n \times h, \quad S_X \in h \times h, \quad V_X \in h \times h$

   In this way $\tilde{K} = U_X S_X^2 U_X^T$ (repeat the same procedure with L and M)

3. Define $\bar{S}_{X_{i,i}} = S^2_{X_{i,i}}, \quad \bar{S}_{Yi,i} = S^2_{Y_{i,i}}, \quad \bar{S}_{Z_{i,i}} = \frac{S^2_{Z_{i,i}}}{S^2_{Z_{i,i}}+\epsilon}$

   and $\bar{G}_X = U_X \bar{S}_X U_X^T, \quad \bar{G}_Y = U_Y \bar{S}_Y U_Y^T, \quad \bar{G}_Z = U_Z \bar{S}_Z U_Z^T$

   We have to fix the parameter ϵ "small enough"

Finally

$$\hat{\mathbb{H}}_{X \perp\!\!\!\perp Y|\mathbf{Z}} = \frac{1}{n^2} tr(\bar{G_X}\bar{G_Y} - 2\bar{G_X}\bar{G_Z}\bar{G_Y} + \bar{G_X}\bar{G_Z}\bar{G_Y}\bar{G_Z}).$$

As in the non-conditional case, we need to calculate the p-value. Three different ways are investigated:

1. Permutation-Cluster test
2. Permutation-Residuals test
3. Residuals test

Permutation - Cluster Test

This method is similar to the permutation method of the non-conditional test. The difference lies in the fact that we have to take into account the conditioning set of variables. In order not to change the marginal distribution of Y given $\mathbf{Z}$ we follow the following steps:

1. Calculate $\mathbb{H}_{X,Y|\mathbf{Z}}$.
2. Cluster $\mathbf{Z}$.
3. Permute p times the elements of Y which fall within the set induced by the clustering of $\mathbf{Z}$.
4. Calculate the p values of $\mathbb{H}_{X \perp\!\!\!\perp Y_{(i)}|\mathbf{Z}}$.
5. Determine under which p-value α the variables are independent calculating the mean value of how many times $\mathbb{H}_{X,Y|\mathbf{Z}} \geq \mathbb{H}_{X,Y_{(i)}|\mathbf{Z}}$.

With this method we have to fix the number of clusters η

Permutation - Residuals Test

Due to the large numbers of parameters to be fixed we introduce the next method which instead of clustering the $\mathbf{Z}$, permutes the residuals of a regression. It consists on the following steps:

1. Calculate $\mathbb{H}_{X,Y|\mathbf{Z}}$.
2. Regress Y on $\mathbf{Z}$ and calculate the residuals r_y.

3. Permute p times the residuals r_y obtaining $\{r_{y(1)}, \ldots, r_{y(p)}\}$ and then calculate $\hat{Y}_i = f(\mathbf{Z}) + r_{y(i)}$, $i \in \{1, \ldots, p\}$.

4. Calculate the p values of $\mathbb{H}_{X \perp\!\!\!\perp \hat{Y}_i | \mathbf{Z}}$.

5. Determine under which p-value α the variables are independent calculating the mean value of how many times $\mathbb{H}_{X \perp\!\!\!\perp Y | \mathbf{Z}} \geq \mathbb{H}_{X, \hat{Y}_{(i)} | \mathbf{Z}}$.

Residuals Test

With this method we test the residuals of the regression to check if the variables are conditionally independent. The steps are:

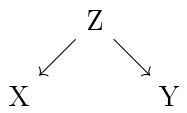
1. Calculate $\mathbb{H}_{X,Y|\mathbf{Z}}$.

2. Regress X on $\mathbf{Z}$ and calculate the residuals r_x.

3. Regress Y on $\mathbf{Z}$ and calculate the residuals r_y.

4. Test whether r_x and r_y are independent with HSIC. We can then use the Permutation or the Gamma test.

Choosing a wrong regression method, we could find dependence when this is not present. A simple example is the following: suppose we have the simple graph shown in Figure 2.11.



where
$Z \sim \mathcal{N}(0, 1)$

$X = Z^2 + \mathcal{U}(-1, +1)$

$Y = Z^2 + \mathcal{G}\text{amma}(1, 2)$.

Figure 2.11: Example Residual test.

We know that $X \perp\!\!\!\perp Y \mid Z$. We then expect to find that the residuals of the regression are independent. If we had used a wrong regression method, for example a linear one as shown in Figures 2.12 and 2.13, the residuals, shown in Figures 2.14 and 2.15, would be dependent, which would lead us to suppose that $X \not\perp\!\!\!\perp Y \mid Z$. If we use an appropriate regression method, this error is avoided. Figures 2.18 and 2.19 show the residuals which seem independent.

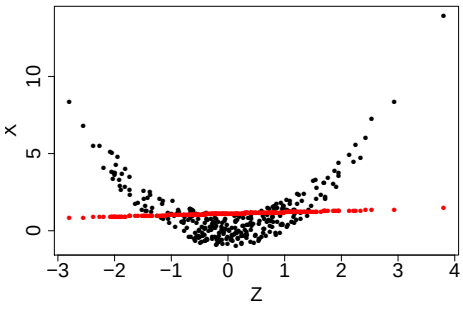


Figure 2.12: In black are represented the sample points, in red are shown the predicted values using a linear regression to fit x given z.

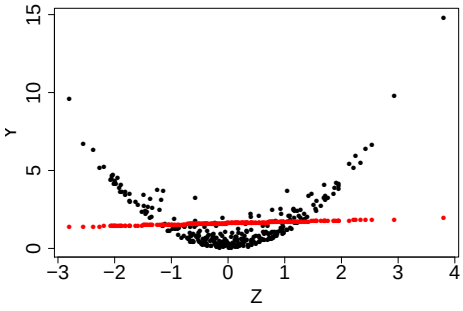


Figure 2.13: In black are represented the sample points, in red are shown the predicted values using a linear regression to fit y given z.

RESIDUALS

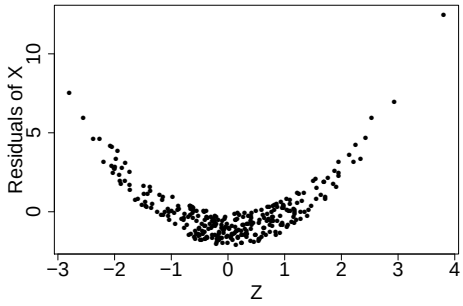


Figure 2.14: The residual of the linear regression.

RESIDUALS

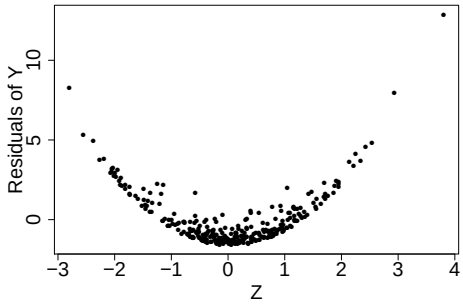


Figure 2.15: The residual of the linear regression.

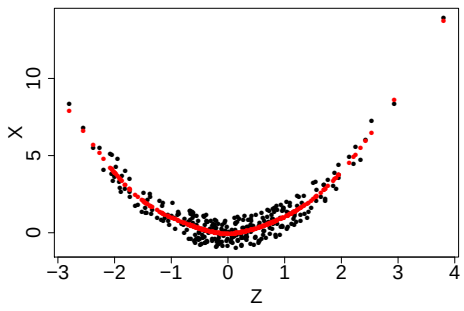


Figure 2.16: In black are represented the sample points, in red are shown the predicted values using a more appropriate regression to fit x given z.

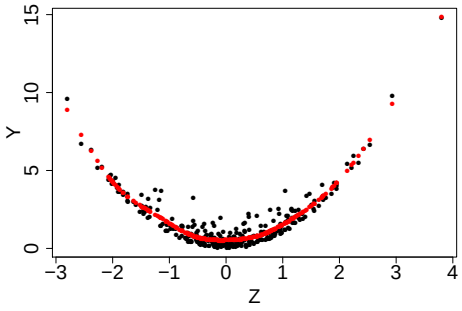


Figure 2.17: In black are represented the sample points, in red are shown the predicted values using a more appropriate regression to fit y given z.

RESIDUALS

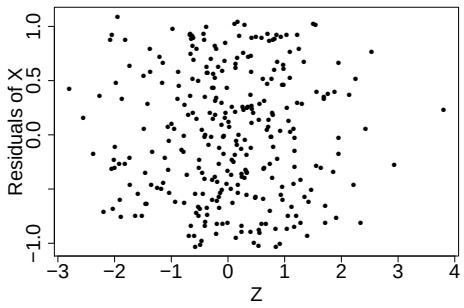


Figure 2.18: The residual of a more appropriate regression.

RESIDUALS

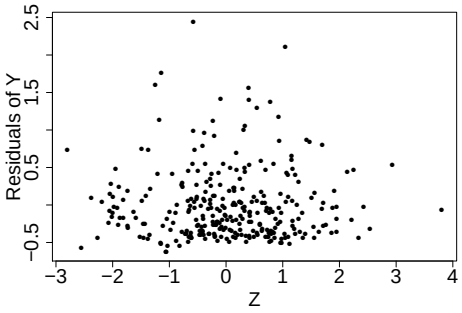


Figure 2.19: The residual of a more appropriate regression.

2.4 Brownian PC

The second idea to solve the problem of finding non-linearity in biological data is to use the Brownian distance correlation. The Brownian distance correlation is a measure of the dependence between variables which may not have the same dimension. This estimator characterizes independence.

From now on, we use the following notations: given a function $g \in \mathbb{R}^{p+q}$ and a weighted function $w(t,s)$, the $\|\cdot\|_w^2$-norm in the weighted L^2 space of the function on $\mathbb{R}^{p+q}$ is defined by

$$\| g(t,s) \|_w = \int_{\mathbb{R}^{p+q}} | g(t,s) |^2 \, w(t,s) dt \, ds.$$

We also assume that $|\cdot|_p$ denotes the Euclidean norm of a vector in $\mathbb{R}^p$.

2.4.1 Brownian Distance Covariance

Let $X \in \mathbb{R}^p$ and $Y \in \mathbb{R}^q$, f_X, f_Y be the distribution of X and Y and $f_{X,Y}$ their joint distribution. Assuming that the random variables X and Y have finite first moments, the Brownian distance covariance is defined as the $\|\cdot\|_w$ - norm of the distance between the product of the density of X and Y and their joint distribution (Székely, Rizzo, Bakirov, et al. 2007; Székely, Rizzo, et al. 2009):

$$\begin{aligned} \nu^2(X,Y;w) &= \| f_{X,Y}(t,s) - f_X(t) f_Y(s) \|_w^2 \\ &= \int_{\mathbb{R}^{p+q}} | f_{X,Y}(t,s) - f_X(t) f_Y(s) |^2 \, w(t,s;\xi) dt \, ds. \end{aligned} \tag{2.2}$$

$w(t,s;\xi)$ is a weighted function such that the Brownian distance correlation is scale invariant and positive for dependent variables. Assuming $\Gamma(\cdot)$ is the complete Gamma function and $C(d,\xi) = \dfrac{2\,\pi^{\frac{d}{2}}\,\Gamma(1-\frac{\xi}{2})}{\xi\, 2^{\xi}\,\Gamma(\frac{\xi+d}{2})}$, a possible choice for $w(t,s;\xi)$ is the following:

$$w(t,s;\xi) = \big(C(p,\xi)C(q,\xi)\,|t|_p^{\xi+p}|\,s\,|_q^{\xi+q}\big)^{-1}, \quad \xi \in (0,2).$$

We will always use the simplest case $\xi = 1$ and thus the weighted function $w(t,s,1)$. To abbreviate the notation we will refer to $w(t,s;1)$ as $w(t,s)$ and to $\nu(X,Y;w)$ as $\nu(X,Y)$.
$\nu^2(X,Y) = 0$ if and only if X and Y are independent. We can also define the Brownian distance correlation $\mathcal{R}(X,Y)$:

$$\mathcal{R}(X,Y) = \begin{cases} \dfrac{\nu^2(X,Y)}{\sqrt{\nu^2(X,X)\nu^2(Y,Y)}}, & \nu^2(X,X)\nu^2(Y,Y) > 0 \\ 0, & \nu^2(X,X)\nu^2(Y,Y) = 0. \end{cases}$$

$\mathcal{R}(X,Y) \in [0,1]$. Given a set of n samples$\{(X_1,Y_1),\ldots,(X_n,Y_n)\}$, we designate:

$$a_{kl} =\mid X_k - X_l \mid_p, \qquad \bar{a}_{k\cdot} = \frac{1}{n}\sum_{l=1}^{n} a_{kl}, \qquad \bar{a}_{\cdot l} = \frac{1}{n}\sum_{k=1}^{n} a_{kl}$$

$$\bar{a}_{\cdot\cdot} = \frac{1}{n^2}\sum_{k,l=1}^{n} a_{k,l} \qquad A_{k,l} = a_{k,l} - \bar{a}_{k\cdot} - \bar{a}_{\cdot l} + \bar{a}_{\cdot\cdot}\ .$$

Similarly, we define

$$B_{k,l} = b_{k,l} - \bar{b}_{k\cdot} - \bar{b}_{\cdot l} + \bar{b}_{\cdot\cdot}$$

for the variable Y.
The empirical distance covariance is defined as

$$\widehat{\nu}^2(X,Y) = \frac{1}{n^2}\sum_{k,l=1}^{n} A_{kl}B_{kl}$$

Thanks to the followings two theorems we can state that $\lim_{n\to\infty}\hat{\nu}(X,Y) = \nu(X,Y)$ almost surely.

Theorem 3. *If* $(\mathbf{X},\mathbf{Y}) = \{(X_1,Y_1),\ldots,(X_n,Y_n)\}$ *is a sample from the joint distribution of (X, Y), then*

$$\widehat{\nu}^2(\mathbf{X},\mathbf{Y}) =\| f^n_{X,Y}(t,s) - f^n_X f^n_Y \|^2_w$$

Where $f^n_X(t) = \frac{1}{n}\sum_{k=1}^{n}\exp\{i\langle t, X_k\rangle\}$, $f^n_Y(s) = \frac{1}{n}\sum_{k=1}^{n}\exp\{i\langle s, Y_k\rangle\}$,

$f^n_{X,Y}(t,s) = \frac{1}{n}\sum_{k=1}^{n}\exp\{i\langle t, X_k\rangle + i\langle s, Y_k\rangle\}$ are the empirical characteristic functions.

Theorem 4. *If* $\mathbb{E}[X] < \infty$ *and* $\mathbb{E}[Y] < \infty$, *then almost surely*

$$\lim_{n\to\infty}\widehat{\nu}(\mathbf{X},\mathbf{Y}) = \nu(X,Y)$$

The reason why the distance covariance is named Brownian is now elucidated. The Brownian motion, known as Wiener process, is a real-valued stochastic process $\{W(t), t \in \mathbb{R}^d\}$ with start in x if (Mörters and Peres 2010):

- $W(0) = x$.
- The process has independent increments, i.e. for all times $t_1 < t_2 < \cdots < t_n$ the increments $W(t_n) - W(t_{n-1})$, $W(t_{n-1}) - W(t_{n-2}), \ldots, W(t_2) - W(t_1)$ are independent.
- $\forall\ t \geqslant 0$ and $\forall\ h > 0$, the increments $W(t+h) - W(t)$ are normally distributed with expectation 0 and variance h.

- Almost surely, the function $t \mapsto W(t)$ is continuous.

We consider two independent Brownian motions $\{W(s),\ s \in \mathbb{R}^p\}$, $\{W'(t),\ t \in \mathbb{R}^q\}$, two random variables $X \in \mathbb{R}^p$, $Y \in \mathbb{R}^q$ with finite second moments and the W-centered version of X:
$X_W = W(X) - \int_{-\infty}^{+\infty} W(s)dF_X(s) = W(X) - \mathbb{E}[W(X) \mid W]$. The Brownian covariance is defined as:

$$W^2(X,Y) = Cov_W^2(X,Y) = \mathbb{E}[X_W X'_W Y_{W'} Y'_{W'}]$$

where the primed variables X' and Y' are and i.i.d. copy of the unprimed X and Y. It turns out that, for arbitrary $X \in \mathbb{R}^p$ and $Y \in \mathbb{R}^q$ with finite second moments:

$$\nu(X,Y) = W(X,Y).$$

2.4.2 Test of Independence for bPC

The independence test is already implemented by `R` in the **`energy`** package (Rizzo and Szekely 2014) with the function **`dcov.test()`**. The test is implemented as a permutation test. The output p-value is calculated as $\frac{1+m}{1+p}$ where m is the number of replicates which are greater than the observed value of the statistic dCov(X,Y) and p is the total number of replicates (personal communication of M.L.Rizzo)

2.4.3 Test of Conditional Independence for bPC

To generalize the test to a conditional version, we use the Residual test explained in 2.3.3. The only difference is that when we test the residuals, we use the Brownian independent test in 2.4.2.

We can briefly summarize the possible tests in this way:

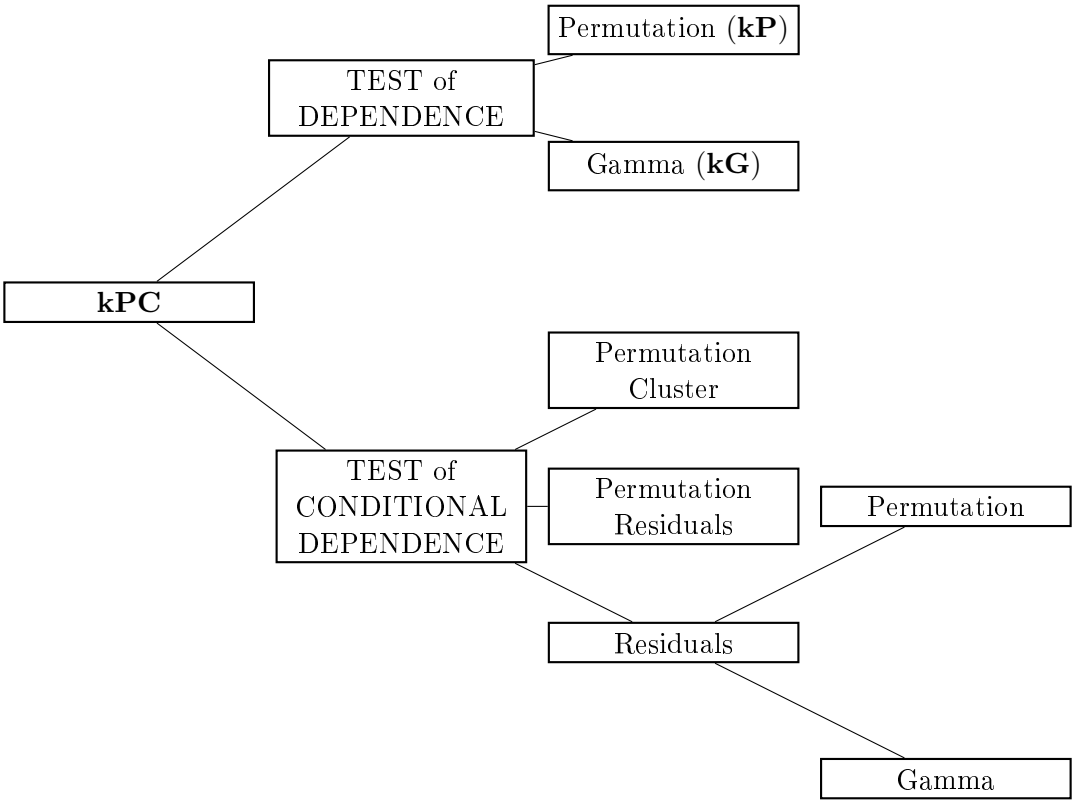


Figure 2.20: Tests in kPC.

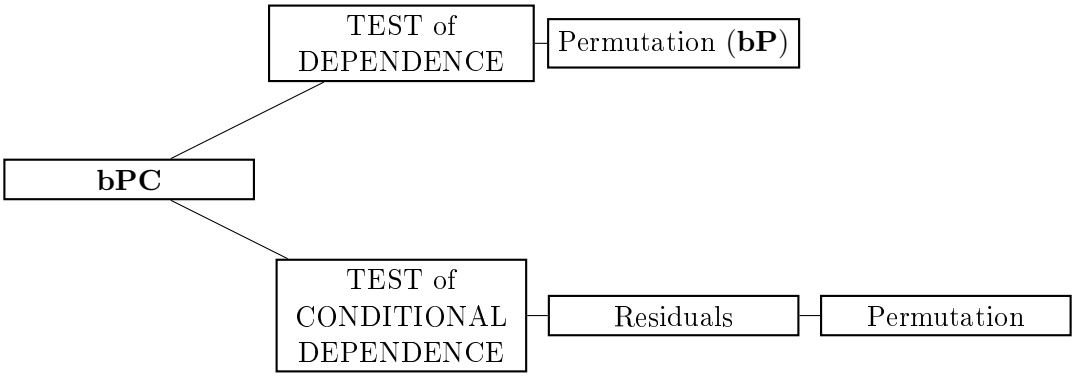


Figure 2.21: Tests in bPC.

2.5 Examples with Simulated Data

Before starting we have to fix three points:

1. The kernel to use in the kPCs .
2. The regression to use in the Residual method (kPCs and bPC) .
3. The clustering to use in kPC Permutation Clustering .

Regarding the point 1, we use a Gaussian kernel, which is a characteristic kernel. The entries of the Gram matrix will be

$$k(x_i, x_j) = \exp(-\frac{\mid x_i - x_j \mid_2^2}{2\sigma}).$$

We have to fix the kernel width σ.
With regard to the point 2, we regress the variables with a Generalized Additive Model (Hastie and Tibshirani 1986). It is a generalization of the General Linear Model. If we are interested in regressing y on the set of variables $x_i, \quad i \in \{1, \ldots, p\}$, with GAM we assume the process to be generated by

$$y = f_0 + \sum_{i=1}^{p} f_i(x_i) + \varepsilon$$

where f_i are unspecified non-parametric functions and ε is an arbitrary noise function. To compute this regression with `R` I use the function `gam()` in the library `mgcv` (Wood 2004; Wood 2011). For the point 3 we use a k-means algorithm, implemented in `R` by the function `kmean()`.

2.5.1 Test of Independence

We will first investigate how well the previous tests of independence work. Figures 2.22-2.25 represent data simulated with non-linear dependences; Figure 2.26 illustrates an example of independence among variables. The number of observations is always three hundred. Table 2.1 shows the corresponding p-values of the tests.

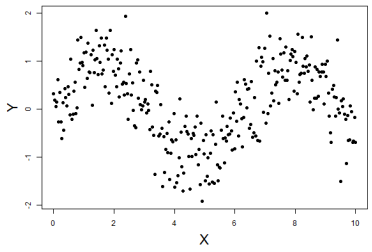


Figure 2.22: $y = \sin(x) + \mathcal{N}(0, 0.25)$.

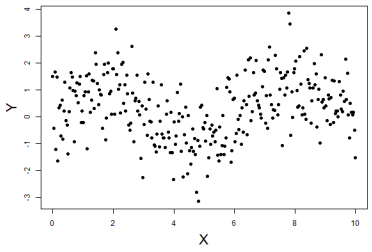


Figure 2.23: $y = \sin(x) + \mathcal{N}(0, 1)$.

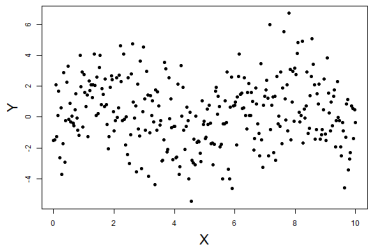


Figure 2.24: $y = \sin(x) + \mathcal{N}(0, 4)$.

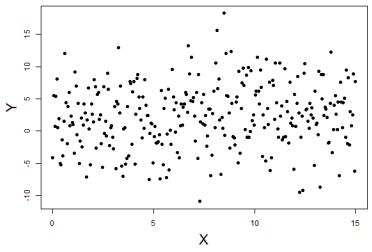


Figure 2.25: $y = \sqrt{x} + \mathcal{N}(0, 25)$.

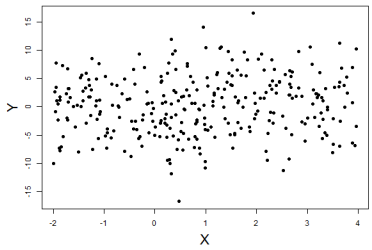


Figure 2.26: $y = \mathcal{N}(0, 25)$ $x = \mathcal{U}(-2, 4)$.

Test	Figure 2.22	Figure 2.23	Figure 2.24	Figure 2.25	Figure 2.26
kP	0	0	6.7e-04	0.24	0.54
kG	0	8.8e-16	9.5e-06	0.24	0.56
bP	9.99e-04	1.2e-03	1.2e-02	6.3e-3	0.35

Table 2.1: Dependence test. kP = kernel based test of dependence with Permutation test. kG = kernel based test of dependence with Gamma test. bP = Brownina test of dependence with Permutation test. All the p-values are a mean of 10 different repetitions of the test. The simulated sample size is 300 for each function.

Figures 2.22 - 2.24 represent the same function with an increasing additive noise. Notice that the p-value is increasing too but is always very low. We can reject the independence. In general, as the noise increases, it becomes more difficult to detect causality. Figure 2.25 shows a pattern which is hardly recognizable. We can see how the two kernel based tests have difficulty in identifying it but the Brownian test succeeds. Finally, as we would expect, the independence is found in Figure 2.26.

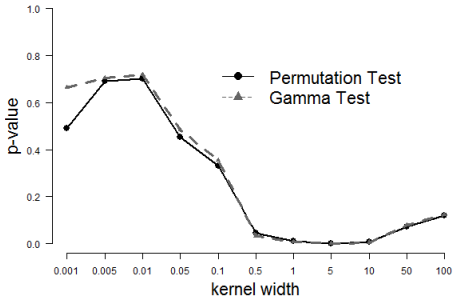


Figure 2.27: $y = \sin(x) + \mathcal{N}(0, 4)$.

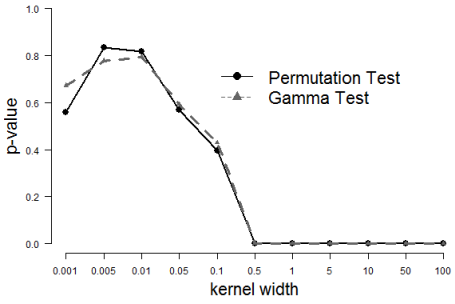


Figure 2.28: $y = \sqrt{x} + \mathcal{N}(0, 25)$.

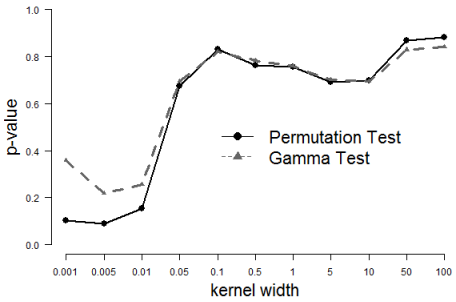


Figure 2.29: $y = \mathcal{N}(0, 25)$ $x = \mathcal{U}(-2, 4)$.

Figures 2.27-2.29 show the p-value (y-axis) of the data illustrated in Figures 2.24-2.26 when $\sigma = \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100\}$ (x-axis). We notice that the Permutation and the Gamma test always give very similar results. We therefore always use the Gamma test since it is computationally more efficient. Moreover, we can see that when the data are dependent (Fig 2.27-2.28), choosing a small σ leads to find independence. This could be due to the fact that, since we use Gaussian kernel, the basis function are too narrow to find dependence. We will focus our attention on $\sigma \in (0.1, 10)$ in order to analyze a big range of values that the p-value can take.

2.5.2 Test of Conditional Independence

In this section we will investigate different ways to detect conditional independence, which characterize each algorithm. Table 2.2 shows which parameters have to be tuned for each algorithm, taking into account that we use

the Gamma test for the dependence test when using HSIC.

kPCs Permutation		kPC Residuals	bPC
kPC Permutation Cluster	kPC Permutation Residuals		
# Permutation p	# Permutation p		# Permutation p
Kernel width σ	Kernel width σ	Kernel width σ	
Regularization parameter ϵ	Regularization parameter ϵ		
# of column in ICD h	# of column in ICD h		
# of clusters η			

Table 2.2: Free parameters

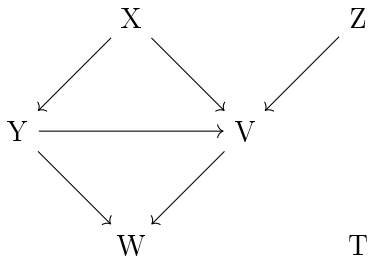


Figure 2.30: Graph.

Analyzing Figure 2.30 where:

$$X \sim \frac{1}{2}\mathcal{G}\text{amma}(2,3) \qquad V = 0.2Y + 0.05X^3 + \cos(Z) + \mathcal{U}(-1,+1)$$

$$Y = \frac{1}{3}X^2 + \mathcal{N}(-2,1) \qquad W = \cos(Y+V) + \mathcal{N}(2,1)$$

$$Z \sim \mathcal{N}(4,1) \qquad T = \mathcal{U}(-1,+1).$$

we illustrate how the regularization parameter ϵ and the number of columns h in the ICD affect the p-value of the test in the kPC Permutation tests. For this purpose the following tests are computed:

Test 1. $X \perp\!\!\!\perp V \mid Y$ (they are dependent, we expect a low p-value).

Test 2. $X \perp\!\!\!\perp Z \mid Y$ (they are independent, we expect a high p-value).

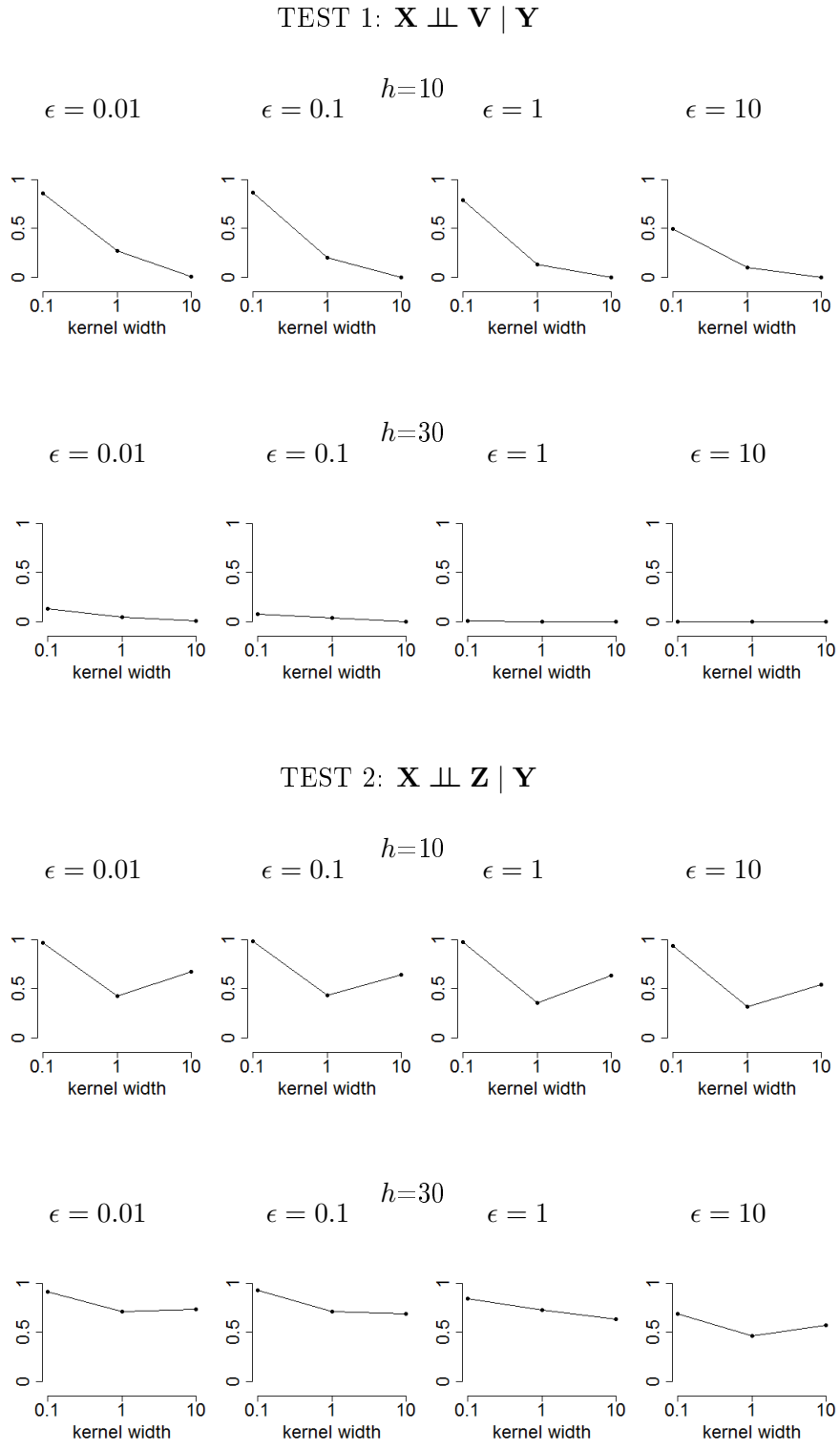


Figure 2.31: In these 16 pictures the p-value(y-axis) is plotted when σ takes the values 0.1, 1, 10. The first 8 refer to the test $X \perp\!\!\!\perp V \mid Y$ (we should find a small p-value), the latter 8 to the test $X \perp\!\!\!\perp Z \mid Y$ (we should find a high p-value). In each of the two blocks, looking at the plot: from left to right, ϵ increases and from the top to the bottom, h increases. The sample data are 100.

When h is low ($= 10$), ϵ influences the value of the p-value more. Nevertheless, for a low computational cost we should use a low h. In order to compensate the loss of information, it would be better to use a high σ. Looking at the smoothness of the curves, we find that the lower h we use, the more instable the result is, because it takes values in a greater range. We have to pay attention not to choose too high ϵ otherwise the effect of the conditioning variables will be removed. As we can see in Table 2.2, when using the kPC Permutation - Cluster and kPC Permutation-Residual tests, lots of parameters must be fixed. The following example illustrates that these algorithms have very similar performances. Additionally, the p-value will be evaluate under different signal to noise ratios (SNR). Given a the relation $Y = X\beta + \epsilon$, where ϵ is the noise, the signal to noise ratio is defined as

$$\text{SNR} = \frac{var(X\beta)}{var(\epsilon)}.$$

We now consider the structures in Figures 2.32 - 2.34. The total amount of data for each function is 200. To perform the kPC Permutations test we fixed the parameters as follows: $\epsilon = 0.1$, $h = 10$, $perm = 150$, $\eta = 4$. We use on purpose three different kinds of basis functions (Fourier, polynomial, piecewise linear). We want to analyze whether the algorithms work better with different kind of functions.

The Figures 2.37, 2.38, 2.41, 2.42, 2.45, 2.46, 2.49, 2.50, 2.53, 2.54, 2.57, 2.58 represent the p-value when σ takes the values 0.1, 0.5, 1, 5, 10. Looking at each pair of Figures 2.37-2.38, 2.41-2.42, 2.45-2.46, 2.49-2.50, 2.53-2.54, 2.57-2.58 we cannot see any relevant differences. We decide to always use the kPC Permutation Residuals. Doing so, we do not have to fix the parameter η. From these pictures it is also clear that we do not have any restriction on σ. We can affirm that, even when the noise increases, the algorithm recognizes that there is (or not) an underlying function. We did not find any set of functions characterized by a basis for which the algorithms were working better or worse. Inspecting other examples as well, it turns out that all the algorithms might be affected by the following errors. When a variable has too much noise, a dependency might not be found, this can influence the subsequent stage. Since the conditioning set is a subset of the adjacent node of one of the two variables we are testing, this leads to test the variables conditioning on not all the sets it should be and to miss the test where two variables would be independent. It could also happen that two nodes are found to be independent before the time due. This means that the two tested nodes are independent given a set S, but the algorithm finds them to be independent given a subset of S. This could cause a misorientation on the arrows. Furthermore, we have discovered that having variables with a big amount of SNR is not always positive. Assume we have a chain structure as in Figure 2.4 where both V_2 and V_3 have high SNR. If we want to test $V_2 \perp\!\!\!\perp V_3 \mid V_1$ we expect to find dependence. V_1 and V_2 describe "in the

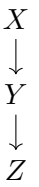


Figure 2.32: Example chain - Fourier basis.

where:

$X = \mathcal{U}(-2, +2)$

$Y = \sin(4X) + \mathcal{N}ormal$

$Z = \cos(7Y) + 2\sin(Y) + \mathcal{U}niform.$

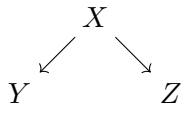


Figure 2.33: Example fork - polynomial basis.

where:

$X = \mathcal{U}(-1, +1.5)$

$Y = \frac{X^2}{6} + \frac{X^6}{10} + \mathcal{N}ormal$

$Z = \frac{2X^3}{3} + X + \mathcal{U}niform.$

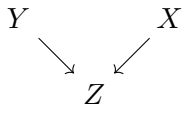


Figure 2.34: Example collider - piecewise constant basis.

where:

$X = \mathcal{U}(-1, +1)$

$Y = \mathcal{U}(-1, 1)$

$$Z = \begin{cases} 0.2 + \mathcal{N}ormal, & XY \in (-\frac{1}{2}, \frac{1}{2}) \\ 0.8 + \mathcal{N}ormal, & XY > 1 \\ 0.45 + \mathcal{N}ormal & XY < -1 \\ \mathcal{N}ormal & else. \end{cases}$$

SNR = 3

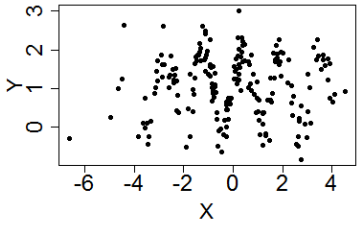


Figure 2.35: Referring to Figure 2.32, it is the plot of X vs Y when SNR = 3.

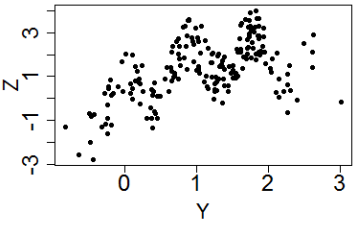


Figure 2.36: Referring to Figure 2.32, it is the plot of Y vs Z when SNR = 3.

TEST: $\mathbf{X} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{Y}$

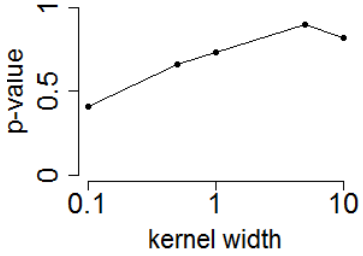


Figure 2.37: p-value of the kPC Permutation Cluster test.

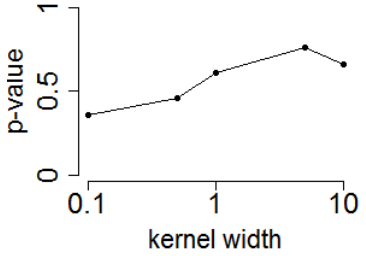


Figure 2.38: p-value of the kPC Permutation Residual test.

SNR = 0.5

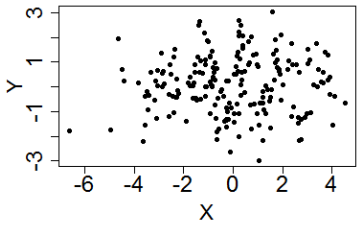


Figure 2.39: Referring to Figure 2.32, it is the plot of X vs Y when SNR = 0.5.

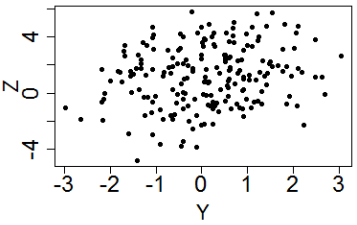


Figure 2.40: Referring to Figure 2.32, it is the plot of Y vs Z when SNR = 0.5.

TEST: $\mathbf{X} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{Y}$

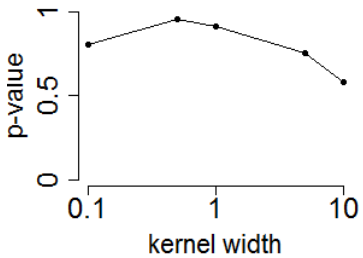


Figure 2.41: p-value of the kPC Permutation Cluster test.

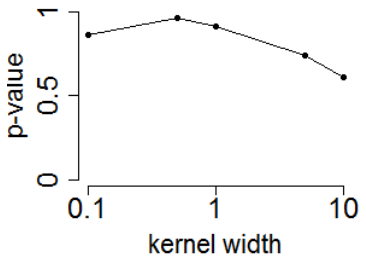


Figure 2.42: p-value of the kPC Permutation Residual test.

SNR = 2

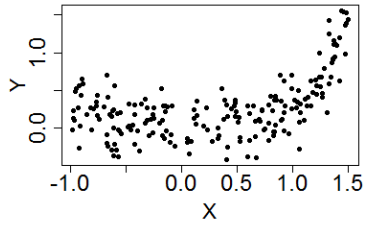


Figure 2.43: Referring to Figure 2.33, it is the plot of X vs Y when SNR = 2.

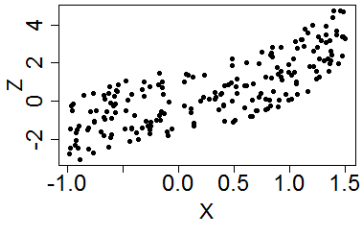


Figure 2.44: Referring to Figure 2.33, it is the plot of X vs Z when SNR = 2.

TEST: $\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$

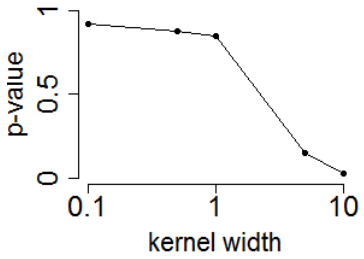


Figure 2.45: p-value of the kPC Permutation Cluster test.

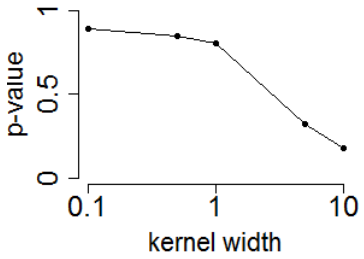


Figure 2.46: p-value of the kPC Permutation Residual test.

SNR = 0.5

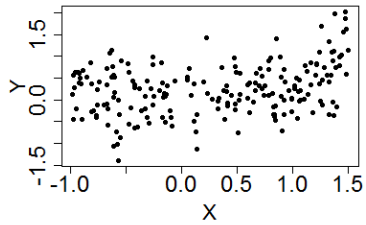


Figure 2.47: Referring to Figure 2.33, it is the plot of X vs Y when SNR = 0.5.

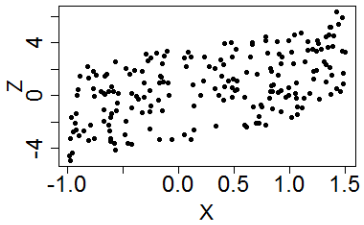


Figure 2.48: Referring to Figure 2.33, it is the plot of X vs Z when SNR = 0.5.

TEST: $\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$

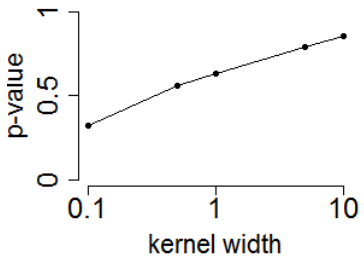


Figure 2.49: p-value of the kPC Permutation Cluster test.

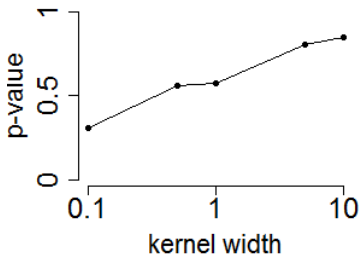


Figure 2.50: p-value of the kPC Permutation Residual test.

SNR = 4

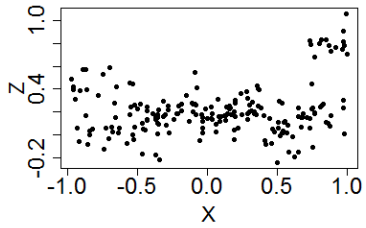


Figure 2.51: Referring to Figure 2.34, it is the plot of X vs Z when SNR = 4.

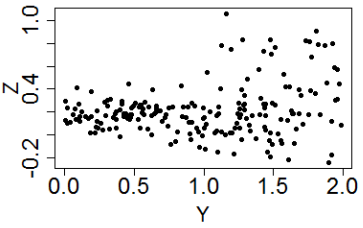


Figure 2.52: Referring to Figure 2.34, it is the plot of Y vs Z when SNR = 4.

TEST: $\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}$

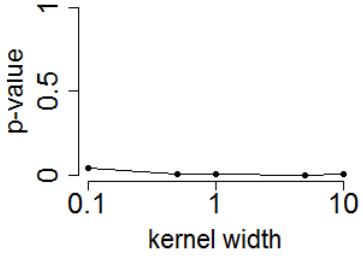


Figure 2.53: p-value of the kPC Permutation Cluster test.

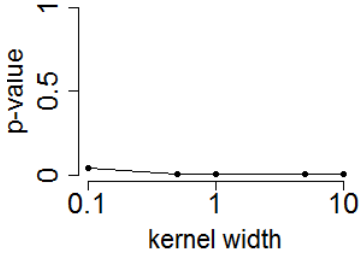


Figure 2.54: p-value of the kPC Permutation Residual test.

SNR = 0.5

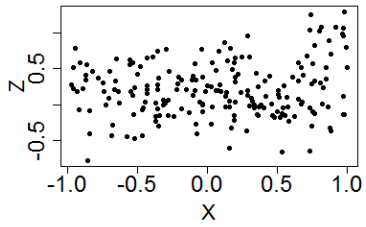


Figure 2.55: Referring to Figure 2.34, it is the plot of X vs Z when SNR = 0.5.

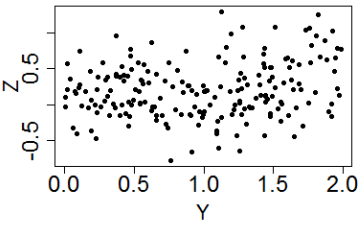


Figure 2.56: Referring to Figure 2.34, it is the plot of Y vs Z when SNR = 0.5.

TEST: $\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}$

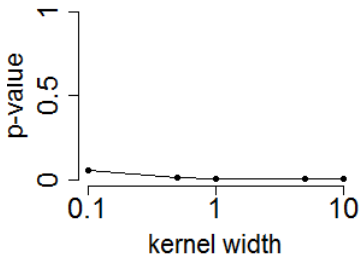


Figure 2.57: p-value of the kPC Permutation Cluster test.

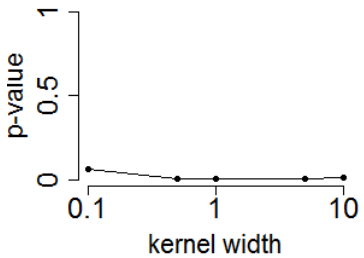


Figure 2.58: p-value of the kPC Permutation Residual test.

same way" V_3 since the relation is almost deterministic, hence the test turns out to be seen as $V_2 \perp\!\!\!\perp V_3 \mid V_2$ which gives independence. Nonetheless we have to keep in mind that the noise is fundamental when we want to detect causal relations, especially when we use residuals methods. As regards the kPCs algorithms, we thought about testing dependence using the value of the HSIC $\mathbb{H}_{XY}$ but we could not identify any threshold to use as a bound to decide whether there was dependence.
We have now restricted the possible algorithms to the following ones:

- kPC Permutation Residuals shortened from now on as kPC Permutation or kPC Permu - Resid
- kPC Residuals
- bPC

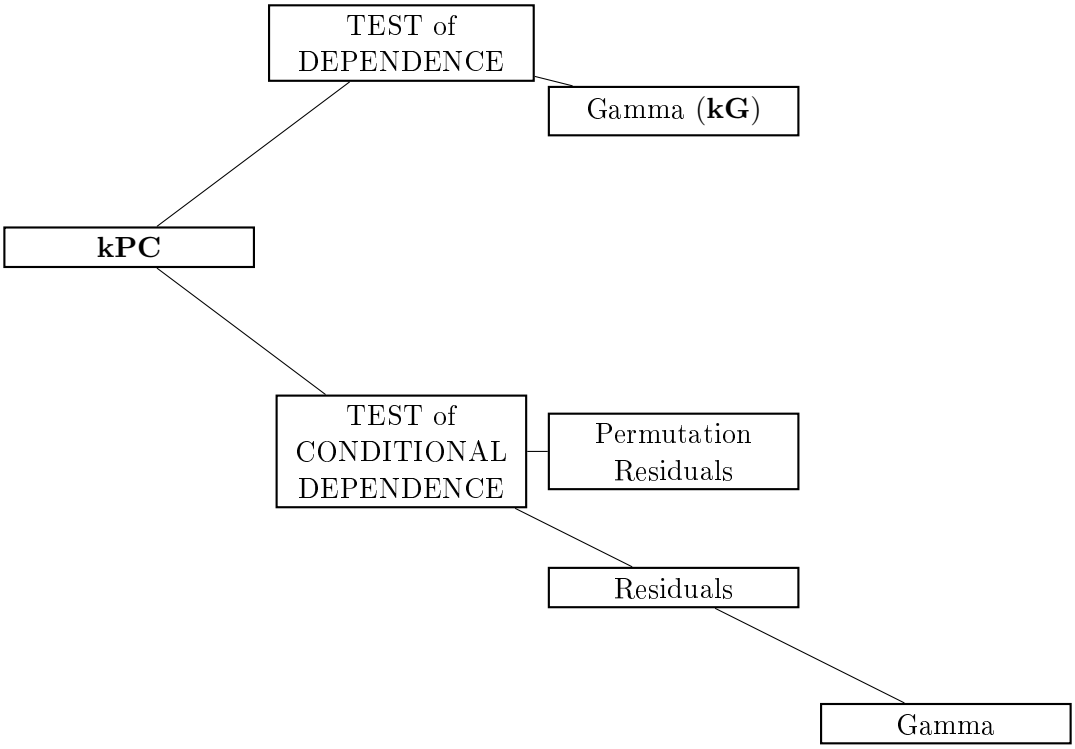


Figure 2.59: Selected tests in kPC.

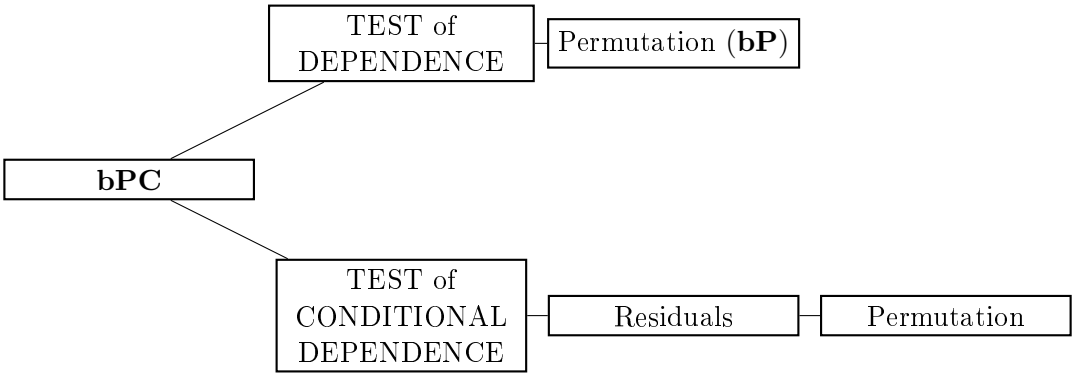


Figure 2.60: Selected tests in bPC.

2.5.3 Comparison with ROC curve

We now illustrate the kPC Permutation Residuals, kPC Residuals, bPC algorithm in an example. To compare the three algorithms with the PC algorithm, we use the ROC curve.
The ROC (Receiver Operating Characteristic) curve is a graphical tool for diagnostic test evaluation. In our purpose it is used to compare the different algorithms. If we know how the data have been generated, so we know the true underlying graphical model, and the output of the test x, for each threshold α it is possible to count the number of:

- True Positives: # of edges which are in the true graph that the test x finds.
- False Positives (error type I): # of edges which are not in the true graph that the test x finds.
- False Negatives (error type II): # of edges which are in the true graph that the test x does not find.
- True Negatives: # of edges which are not in the true graph that the test x does not find.

These quantities can be represented in the Contingency Table as shown in Table 2.3.

		TRUE CONDITION	
		Positive	Negative
TEST OUTCOME	Positive	True Positive **TP**	False Positive **FP**
	Negative	False Negative **FN**	True Negative **TN**

Table 2.3: Contingency table.

The sensitivity and specificity are defined as

$$\textit{Sensitivity} = \frac{TP}{TP + FN}$$

$$\textit{Specificity} = \frac{TN}{TN + FP}.$$

They both take values between 0 and 1. The sensitivity evaluates the ability of the test in identifying the correct dependencies. It is also called True positive rate. The specificity explains the ability of the test in not finding dependencies which do not exist. A perfect test would have both parameters

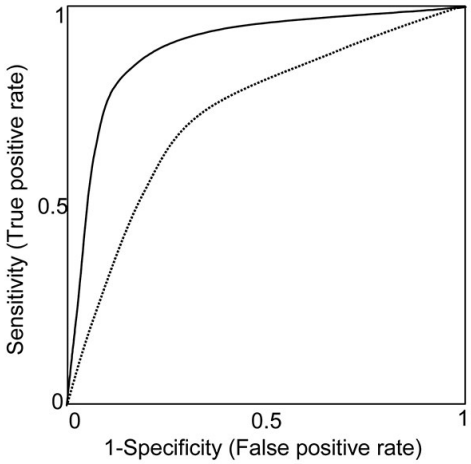


Figure 2.61: ROC curve.

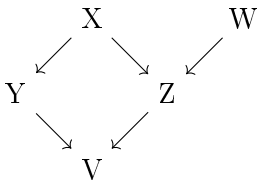


Figure 2.62: Example 5 variables.

where

$$X = \mathcal{N}(0,1)$$

$$Y = X^2 + \mathcal{G}\text{amma}(2,2)$$

$$Z = \sin(2W) + \lceil X \rceil^2 + \mathcal{U}(-1,1)$$

$$V = \cos(Z) + 3\sin(Y) + \mathcal{U}(0,2)$$

$$W = \mathcal{U}(-2,5)$$

high, but this is almost impossible. In order to have *Sensitivity* $= 1$, we should have an *Error Type II* $= 0$ which always means to refuse the independence. But this would lead to have *Specificity* $= 0$ because $TN = 0$. The ROC curve (Figure 2.61) represents values of $1 -$ *Specificity*, known also as False positive rate versus *Sensitivity* at different α levels. The higher α is, the more the independence will be refused and *Sensitivity* and $1 -$ *Specificity* will be close to 1. The closer the curve is to the left and top edges, the better is the test. A numerical estimator for the goodness of the ROC curve is the AUC, Area Under the Curve.

If we consider the graph illustrated in Figure 2.62, the correct edges are 5 and the possible edges are 10 in total. Looking at the Figure 2.63, we can see that the methods kPCs and bPC have high performances. This is noticeable analyzing the AUC, which is always higher than 0.95. Moreover, the kPC Permutation is able to reconstruct the true underlying graph when $\alpha \in (0.44; 0.46)$.

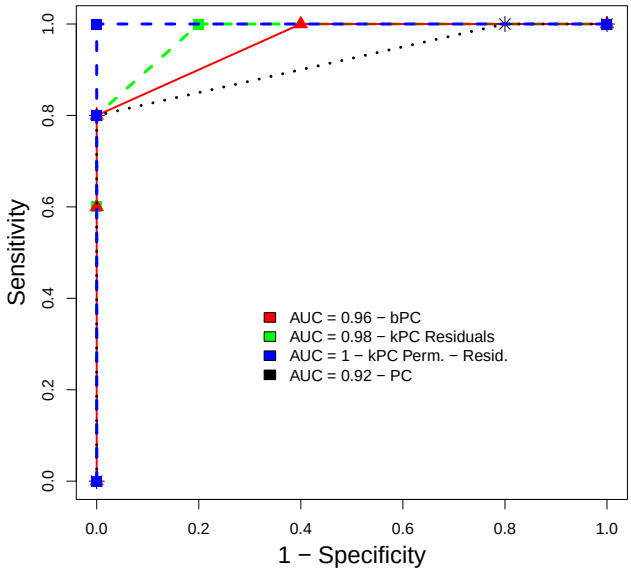


Figure 2.63: ROC curve for kPC, bPC and PC algorithms.

Comparing the three algorithms with the PC, we notice that PC has performances lower than the others. In order it succeeds in finding the five correct connections, it first finds 4 wrong connections, as we can see by the point in (0.8; 1).

2.6 Generalization of Transitive phase

We now focus on restricting the output class of possible DAGs which describe the causal relationship among the data. The non-linearity and non-Gaussianity of the data are helpful features that lead us to discover more causal orientations. This is due to the fact that non-linearity breaks the symmetry among the observed variables and allows the identification of the causal directions.

2.6.1 Additive Noise model

When we define the data with a generalized additive model, we suppose the model be generated as follow (Hoyer et al. 2009):

$$V_i = f(Pa_{\mathbf{G}}(V_i)) + \varepsilon_i, \qquad i \in \{1, \ldots, n\}$$

where:

- $\mathbf{G}$ is the DAG we want to represent.
- V_i are the variables.
- $f(\cdot)$ is an arbitrary function (it could be non-linear).
- ε_i are the noise functions, which have an arbitrary density and are jointly independent.

Assuming we have two variables X and Y which are dependent, we are interested in discovering what is the direction of the causal relation. The suggested approach is the following:

1. non-linearly regress X on Y: $X = f(Y) + \varepsilon_Y$ (backward model) and calculate the residuals $\hat{\varepsilon}_X$.
2. non-linearly regress Y on X: $Y = f(X) + \varepsilon_X$ (forward model) and calculate the residuals $\hat{\varepsilon}_Y$.
3. test the dependence of X and $\hat{\varepsilon}_Y$.
4. test the dependence of Y and $\hat{\varepsilon}_X$.

If both tests in 3 and 4 find independence, we cannot infer the direction by those data. If both find dependence this model is not good enough to describe the data. Finally, if we find in one test dependence and in the other independence, we orient the causal relation in the direction where the independence is found. We consider this example where the function are non-linear and the noise is not Gaussian. Figures 2.64 and 2.65 show the

$$\begin{aligned} X &= \varepsilon_X & \varepsilon_X &\sim \mathcal{U}(-1,+1) \\ Y &= X^2 + \varepsilon_Y & \varepsilon_Y &\sim \mathcal{U}(-1,+1) \end{aligned}$$

forward model $Y = f(X) + \varepsilon_Y$ and the backward model $X = f(Y) + \varepsilon_X$ with the predicted points of the regression. Analyzing the residuals of the regression, see Figures 2.66 and 2.67, we notice that there is independence in the first figure but not in the second. This leads us to conclude that the model $Y = f(X) + \varepsilon_Y$ is consistent with the data, which is correct.
On the other hand, if the data were linear and Gaussian, it would have been impossible to detect the direction of the causal relationship. In both cases we would have independence, as illustrated in the following example:

$$X = \varepsilon_X \qquad \varepsilon_X \sim \mathcal{N}(0,1)$$
$$Y = 3X + \varepsilon_Y \qquad \varepsilon_Y \sim \mathcal{N}(0,1)$$

In both forward and backward models, the residuals result be independent and we cannot infer the direction of the edge.

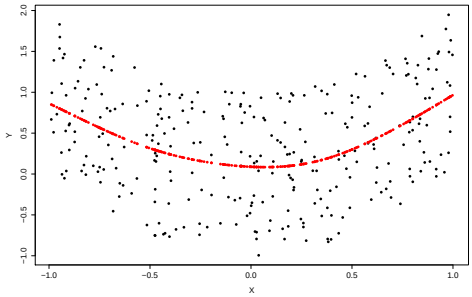


Figure 2.64: In black are represented the sample points, in red are shown the predicted values.

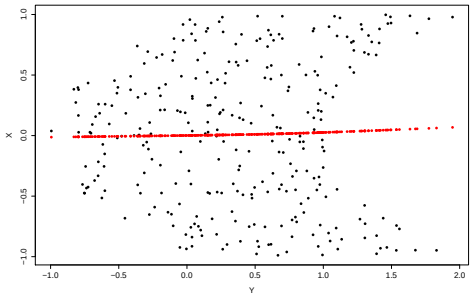


Figure 2.65: In black are represented the sample points, in red are shown the predicted values.

RESIDUALS

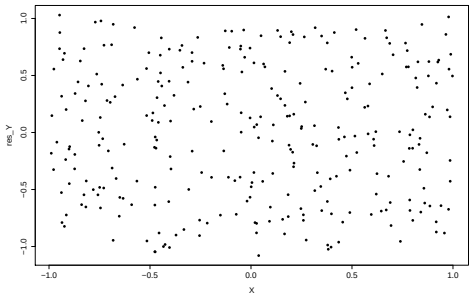


Figure 2.66: The residuals of the regression $y = f(x) + \varepsilon$.

RESIDUALS

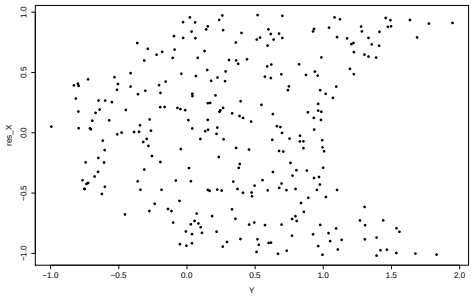


Figure 2.67: The residuals of the regression $x = f(y) + \varepsilon$.

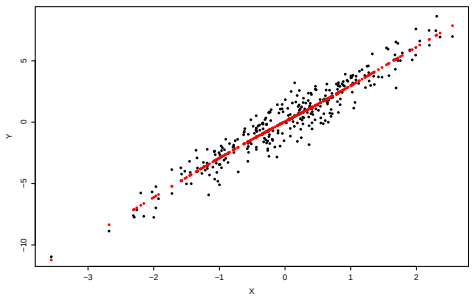


Figure 2.68: In black are represented the sample points, in red are shown the predicted values.

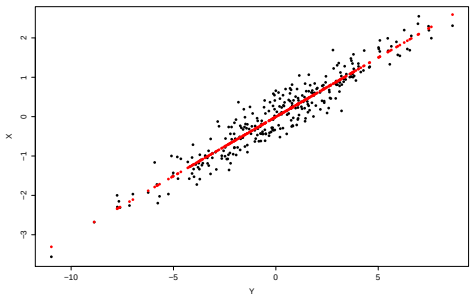


Figure 2.69: In black are represented the sample points, in red are shown the predicted values.

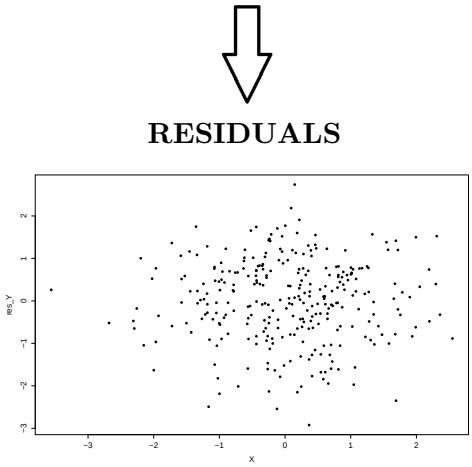


Figure 2.70: The residuals of the regression $y = f(x) + \varepsilon$.

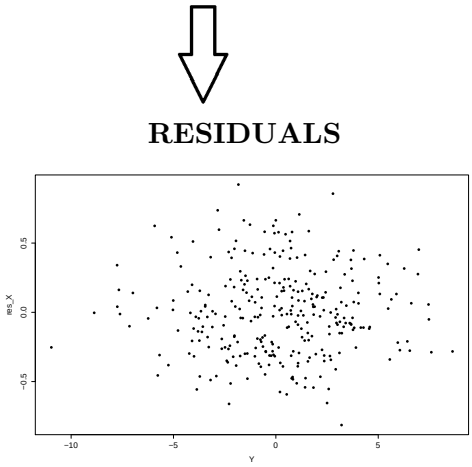


Figure 2.71: The residuals of the regression $x = f(y) + \varepsilon$.

2.6.2 Weakly Additive Noise model

Even thought the previous procedure helps in finding more causal directions among variables comparing to the Transitive step of PC, it suffers from two problems (Tillman 2009):

- In some settings the additive noise model is invertible: $X \perp\!\!\!\perp \hat{\varepsilon}_Y$ and $Y \perp\!\!\!\perp \hat{\varepsilon}_X$. We cannot infer the direction of the causality.
- The additive noise model is not closed under the marginalization of intermediary variables when $f(\cdot)$ is non-linear. When the model is $X \to Y \to Z$ but the variable Y in not measured the model is seen as $X \to Z$ but it can happen that $X \not\perp\!\!\!\perp \hat{\varepsilon}_Z$ and $Z \not\perp\!\!\!\perp \hat{\varepsilon}_X$. We cannot infer the direction of the causality.

- It is mandatory that the noise is additive.

To overcome those problems, the weakly additive noise model is introduced (Tillman, Gretton, and Spirtes 2009).

Definition 2. *$\phi = \langle V_i, \mathrm{Pa}_{\mathbf{G}}(V_i)\rangle$ is a local additive noise model for a distribution P over the nodes $\mathbf{V}$ that is Markov to a DAG $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}\rangle$ if $V_i = f(\mathrm{Pa}_{\mathbf{G}}(V_i)) + \varepsilon_i$ is an additive noise model.*

Definition 3. *A weakly additive noise model $\mathcal{M} = \langle \mathbf{G}, \mathbf{\Phi}\rangle$ for a distribution P over $\mathbf{V}$ is a DAG $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}\rangle$ and a set ϕ of local additive noise models such that:*

1. *P is Markov to $\mathbf{G}$.*
2. *$\phi \in \Phi$ if and only if ϕ is a local additive noise model for P contained in $\mathbf{G}$.*
3. *$\forall \langle V_i, \mathrm{Pa}_{\mathbf{G}}(V_i)\rangle$, there does not exist a $V_j \in \mathrm{Pa}_{\mathbf{G}}(V_i)$ and an arbitrary directed graph $\mathbf{G}'$ (not necessary related to P) such that $V_i \in \mathrm{Pa}_{\mathbf{G}'}(V_j)$ and $\langle V_j, \mathrm{Pa}_{\mathbf{G}}(V_j)\rangle$ is a local additive noise model for P contained in $\mathbf{G}'$.*

Assuming the data are generated by a weakly additive noise model means that if the real underlying structure of the data is $X \to Y$, there exists no functional form such as the data can be represented as $X = f(Y) + \varepsilon$ and $\varepsilon_X \perp\!\!\!\perp Y$

Definition 4. *A weakly additive noise model $\mathcal{M} = \langle \mathbf{G}, \mathbf{\Phi}\rangle$ is distribution equivalent to $\mathcal{N} = \langle \mathbf{G}', \mathbf{\Phi}'\rangle$ if and only if*

1. *$\mathbf{G}$ and $\mathbf{G}'$ are Markov equivalent.*
2. *$\phi \in \Phi$ if and only if $\phi \in \Phi'$.*

Definition 5. *A weakly additive noise partially directed acyclic graph (WANPDAG) for $\mathcal{M} = \langle \mathbf{G}, \mathbf{\Phi}\rangle$ is a mixed graph $\mathbf{H} = \langle \mathbf{V}, \mathbf{E}\rangle$ such that for $\{V_i, V_j\} \subseteq \mathbf{V}$:*

1. *$V_i \to V_j$ is a directed edge in $\mathbf{H}$ if and only if $V_i \to V_j$ is a directed edge in $\mathbf{G}$ and in all $\mathbf{G}'$ such that $\mathcal{N} = \langle \mathbf{G}', \mathbf{\Phi}'\rangle$ is distribution equivalent to $\mathcal{M}$.*
2. *$V_i - V_j$ is an undirected edge in $\mathcal{H}$ if and only if $V_i \to V_j$ is a directed edge in $\mathcal{G}$ and there exists a $\mathcal{G}'$ and $\mathcal{N} = \langle \mathbf{G}', \mathbf{\Phi}'\rangle$ distribution equivalent to $\mathcal{M}$ such that $V_i \leftarrow V_j$ is a directed edge in $\mathbf{G}'$.*

The following algorithm is used instead of the Transitive step in order to find more orientations. It is a generalization of the Transitive step because, in order to find the causal dependencies, it inspects the residuals of the regressions and after it uses the Transitive step to collect more informations. The underlying idea to learn more about causality is the generalization of the two-variables case explicated in Section 2.6.1. It takes into account the fact that we want to test the dependence of two variables, each of them connected, directly or not, to other variables. In order to avoid the complications arising with the additive noise model, the algorithm is structured such that the output is a WAN-PDAG. Defining $U_G^{V_i}$ as the set of all nodes adjacent to V_i by an undirect edge, the algorithm is the following:

Algorithm 4 GENERALIZED TRANSITIVE phase

Input: G
Output: G

$s = 1$
while $\max_{V_i \in V} \mid U_G^{V_i} \mid \geq s$ **do**
  for all $V_i \in V$ such as $\mid U_G^{V_i} \mid = s$ or $\mid U_G^{V_i} \mid < s$ and $U_G^{V_i}$ was updated **do**
    $s' = s$
    while $s' > 0$ **do**
      for all $S \subseteq U_G^{V_i}$ such that $\mid S \mid = s'$ and $\forall S_k \in S$, orienting $S_k \to V_i$, does not create immorality **do**
        non-parametrically regress V_i on $Pa_G^{V_i} \cup S$ and compute the residual $\hat{\varepsilon}_{iS}$
        if $\hat{\varepsilon}_{iS} \perp\!\!\!\perp S$ and $\nexists V_j \in S$ and $S' \subseteq U_G^{V_j}$ such that regressing V_j on $Pa_G^{V_j} \cup S' \cup V_i$ results in the residual $\hat{\varepsilon}_{jS \cup V_i} \perp\!\!\!\perp S' \cup V_i$ **then**
          $\forall S_k \in S$, orient $S_k \to V_i$ and $\forall U_l \in U_G^{V_i} \backslash S$ orient $V_i \to U_l$
          Apply the Transitive step
          $\forall V_m \in V$ update $U_G^{V_m}$, set $s' = 1$ and break
        end if
      end for
      $s' = s' - 1$;
    end while
  end for
  $s = s + 1$;
end while

Using the Generalized transitive step there are less chances that an edge oriented $\leftrightarrow$ in the Collider step, is reoriented as $\to$ or $\leftarrow$ in the last step. This algorithm does not totally exclude the reorientation to happen. In order to avoid this problem I will use a modified Transitive step, which forbids to modify double oriented edges. I will refer to it as Transitive step and always

use it.

3 APPLICATION TO REAL DATA

In this chapter the three selected algorithms and the Generalization of the transitive step will be applied (i) to a real dataset and (ii) to data simulated from the real data. We use simulated data from the real one in order to have a ground truth to use when comparing the algorithms The goal is to evaluate if the found methods are suitable to depict highly non-linear patterns and thus to compare bPC and kPCs with the PC algorithm.

3.1 Data Analysis

Proteins are signaling molecules which receive and send signals. The nature of those signals can be chemical, mechanical and so on. In a healthy cell, as a molecule receives a signal, it undergoes some responses and transformations, which might lead in their turn to triggering other modifications inside of the cell. As a second protein is affected by internal changes, a pathway can be detected, and so on until the end of the propagation. Pathways can involve lots of proteins. Graphical models are used to describe those causal protein-signaling networks; they investigate the flow of information from one protein to another.
The datasets which will be analyzed are the ones published by Sachs et al. 2005. There are eight experimental datasets. Each dataset reports the value of eleven protein's levels expression. The measured proteins are RAF, MEK, ERK (aka P44.42), PLCγ, PIP2, PIP3, PKC, AKT, PKA, JNK, P38. The number of observations varies from one dataset to another, but it is always around 700-900. Every dataset is characterized by the quantitative value of the protein expression after a series of specific stimulatory cues or inhibitory interventions. Table 3.1 illustrates the interventions introduced for each dataset.
The method used to capture the protein's levels expression is the *Intracellular Multicolor Flow Cytometry*. It allows the quantitative measurement of multiple proteins states and modifications in many thousands of individual

DATASET 1	General perturbation	
DATASET 2	General perturbation	
DATASET 3	Activation	PKA
DATASET 4	Inhibition	AKT
DATASET 5	Inhibition	MEK1
DATASET 6	Activation	PKC
DATASET 7	Inhibition	PKC
DATASET 8	Inhibition	PIP2

Table 3.1: Table of datasets: for each dataset is explained after which kind of external perturbation the proteins have been measured.

cells. In the experiment performed to obtain the data we deal with, the flow cytometry technique is used to measure modification states of proteins, such as phosphorylation trough antibodies. A protein is phosphorylated when a phosphate group PO_4^{3-} is added to it. The reasons why this technique is adopted are:

- There are not any average among the population. The measurement occur within every cell.
- Every cell represents an independent observation.
- The measurements occur in single cells: for each experiment thousands of data are available.

Obviously there is also a problem, which is that by now there are only about 80 antibodies compatible with this method. Scientist are faithful that this number will increase. A situation which comes from this problem is the following. PKC phosphorylates RAF at S497, S499, S259, but in the experiment are only used antibodies which detect RAF S259. Hence, it can happen that a dependency is not seen or that a pathway excludes a protein because the phosphorylated protein is not detected.
In literature there exist many "well-know" pathways, which are illustrated in Figure 3.1 and summarized in Figure 3.2 (green edges). There are also other connections, which are cited in fewer books of biology. Those are:

- ERK $\rightarrow$ AKT
- PKC $\rightarrow$ PKA
- PKA $\rightarrow$ MEK

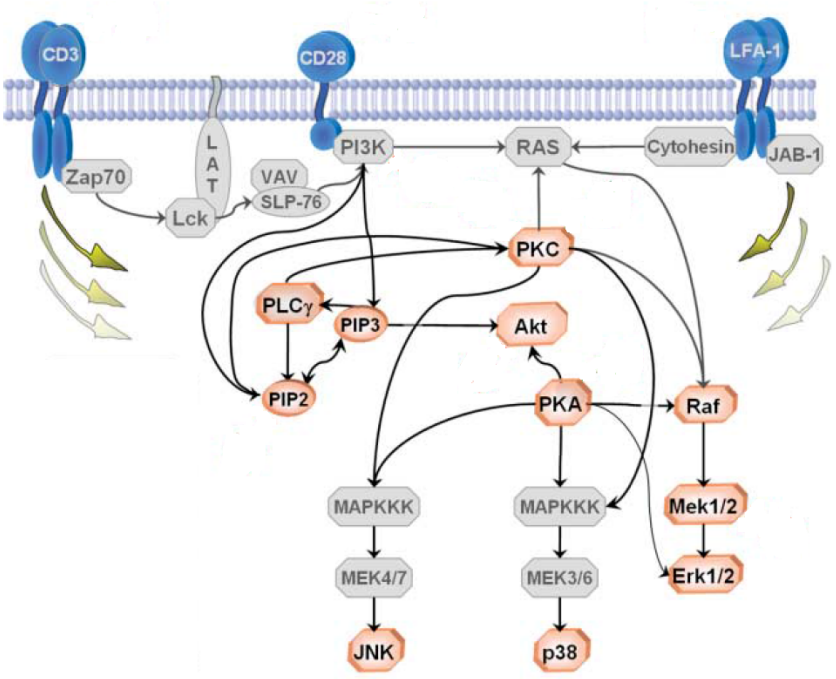


Figure 3.1: "Well known dependences" among variable which are reported in many books of biology. The red proteins are the measured ones.

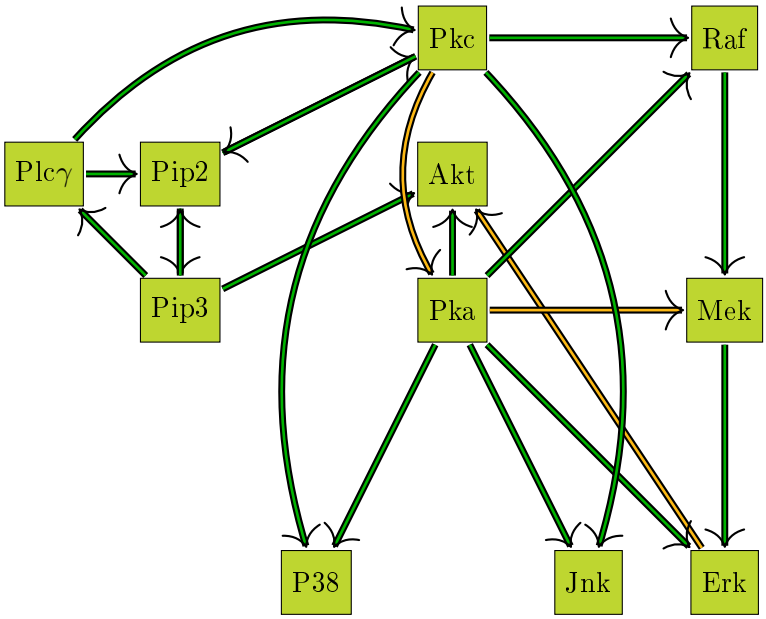


Figure 3.2: Summary of the the "well-known dependencies" (green) and "known dependencies" (orange).

These are represented in Figure 3.2 with orange edges. In the following pages I will refer to them as "known dependencies". As we notice, it will be difficult for the algorithms to find the ground truth to use to compare with the outputs of the algorithms. First, because not all causal relationships have been established. No one can guarantee us that in future other dependences among those proteins will not be found. Secondly, the ground truth as illustrated in Figure 3.1, is made by cyclic pathways, which are not recognizable by the algorithms. We also notice that the direct relationships the algorithms must find are not direct in nature, because not all the proteins in Figure 3.1 have been recorded.
The analyzed data are the log-transformation of the raw data. For sake of simplicity, we only show how one of the dataset looks like after the transformation. Figures 3.3 - 3.4 show the data in dataset 8. We observe that some dependencies are clear and seems linear, as RAF-MEK. On the other hand, others are less noticeable, such as RAF-PKC. This trend of finding some dependencies visible while others are not, is noticeable in all the 8 datasets. Comparing the value of a single protein among each dataset, we find it to be very different from one dataset to another. This is understandable because in some dataset the variable is externally activated or modified by the phosphorylation of another molecule. But it can also happen that in other datasets the protein is not affected by any interventions.

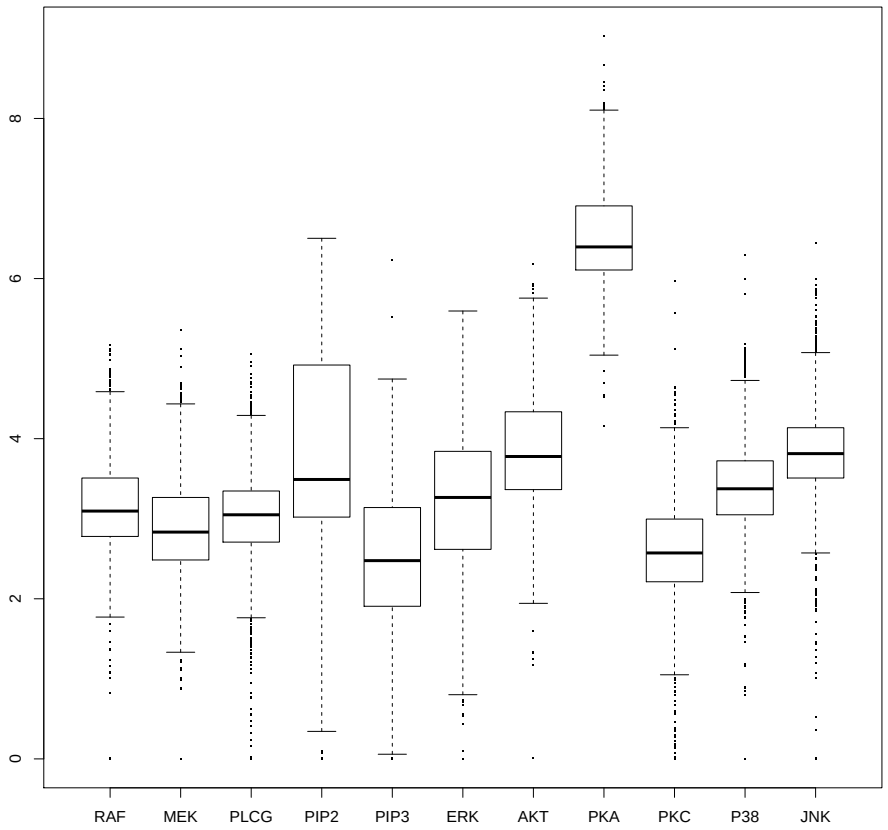


Figure 3.3: Boxplot of the data in dataset 8 after the log - transformation.

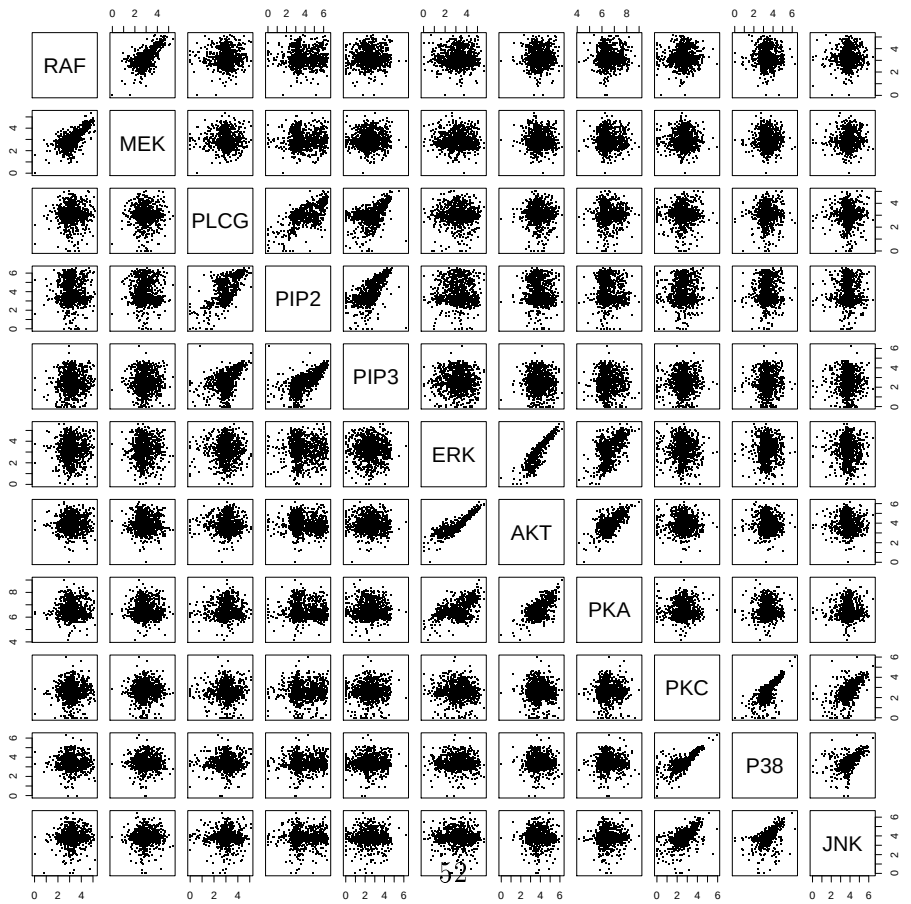


Figure 3.4: Plot of the data in dataset 8 after the log - transformation.

3.2 Data simulation

In order to compare the efficiency of the algorithms when the measurement is affected by decreasing noise, we use data simulated from the original data. The idea is to consider a certain network as the ground truth and to generate data which satisfy the links in the ground truth. It is important to simulate the relationships among variables as close as possible to the real ones. For each dataset the simulation is different, because it takes into account that a certain protein has been externally modified. If we consider dataset 8, the inhibited protein is PIP2. PIP2 is externally modified and we suppose it cannot be perturbed by any protein in the network. We expect to simulate a causal model as in Figure 3.5. The data generation starts from the nodes which do not have parents: in this case, obviously PIP2 and also PKA. We will refer to them as "starting nodes". The value of these variables is the same as in the real dataset, because we cannot infer their values from other sources. That causes that when the data will be generated with decreased noise, those variables will not have reduced noise. The next step is to generate the nodes for which their parents have either already been simulated or are starting nodes. With respect to Figure 3.5 we simulate the values of the protein

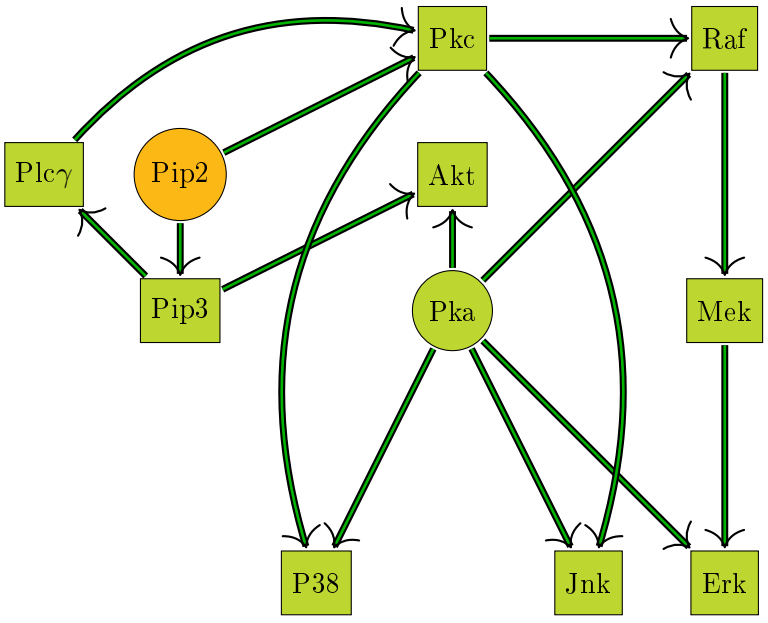


Figure 3.5: Simulated graph when the protein PIP2 (orange box) is externally modified. The proteins in circled boxes are the starting nodes.

expressions in this order:

I.	PIP3	from	PIP2		
II.	PLCγ	from	PIP3		
III.	PKC	from	PIP2	and	PLCγ
IV.	AKT	from	PIP3	and	PKA
V.	RAF	from	PKC	and	PKA
VI.	MEK	from	RAF		
VII.	ERK	from	MEK	and	PKA
VIII.	P38	from	PKC	and	PKA
IX.	JNK	from	PKC	and	PKA

The simulation is implemented as follow:

1. Regress the real value of the node V_i on its Parents Pa^{V_i} with a Generalized Additive Model.
2. Calculate the vector r_i of residuals of the regression, permute it randomly obtaining $r_{(i)}$. If we want data with less noise, reduce the residuals dividing them by k: $\frac{r_{(i)}}{k}$.
3. Calculate the predicted values of the regression $\widehat{V}_i$.
4. The simulated values of the variable V_i are:

$$\widehat{V}_i + \frac{r_{(i)}}{k}.$$

Figures 3.6 and 3.7 illustrate the values of the simulated dataset 8 with non-decreased residuals. Comparing it with Figure 3.3 and 3.4 we cannot find any large differences. This remark applies to all other simulated datasets with non-reduced residuals. As we reduce the residuals, all but the starting node variables take values in a narrower range. In the following pages we will use simulated data with non-reduced residuals and residuals lowered by a factor 3 and 10. We will refer to them as "simulated data - residuals/1", "simulated data - residuals/3" and "simulated data - residuals/10".

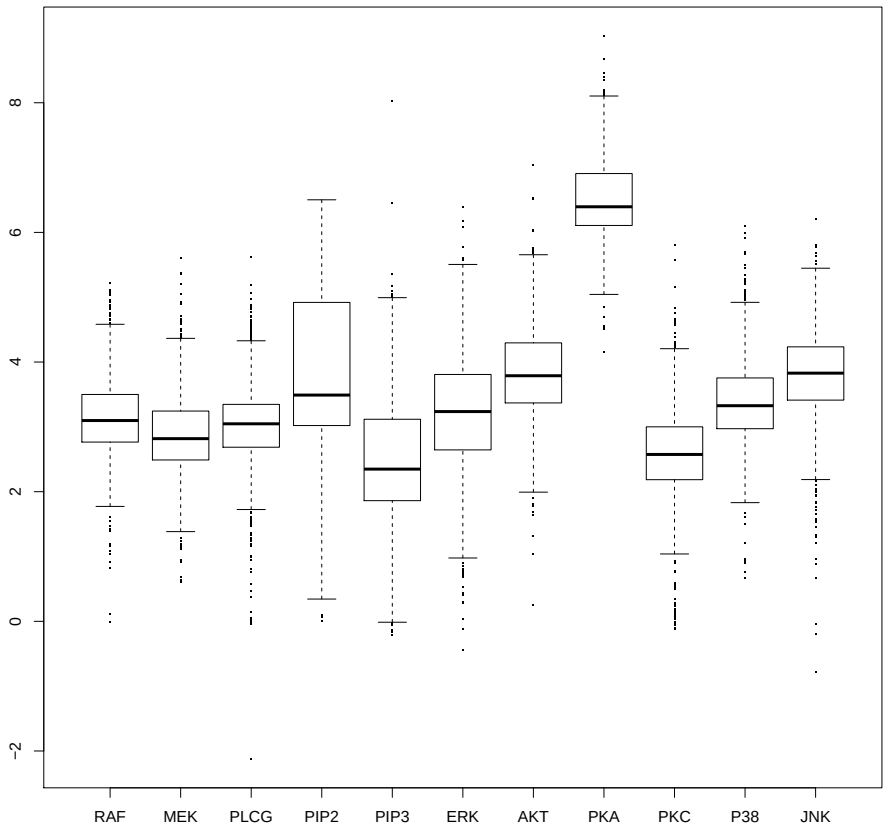


Figure 3.6: Boxplot of the Simulate dataset 8 - residuals/1.

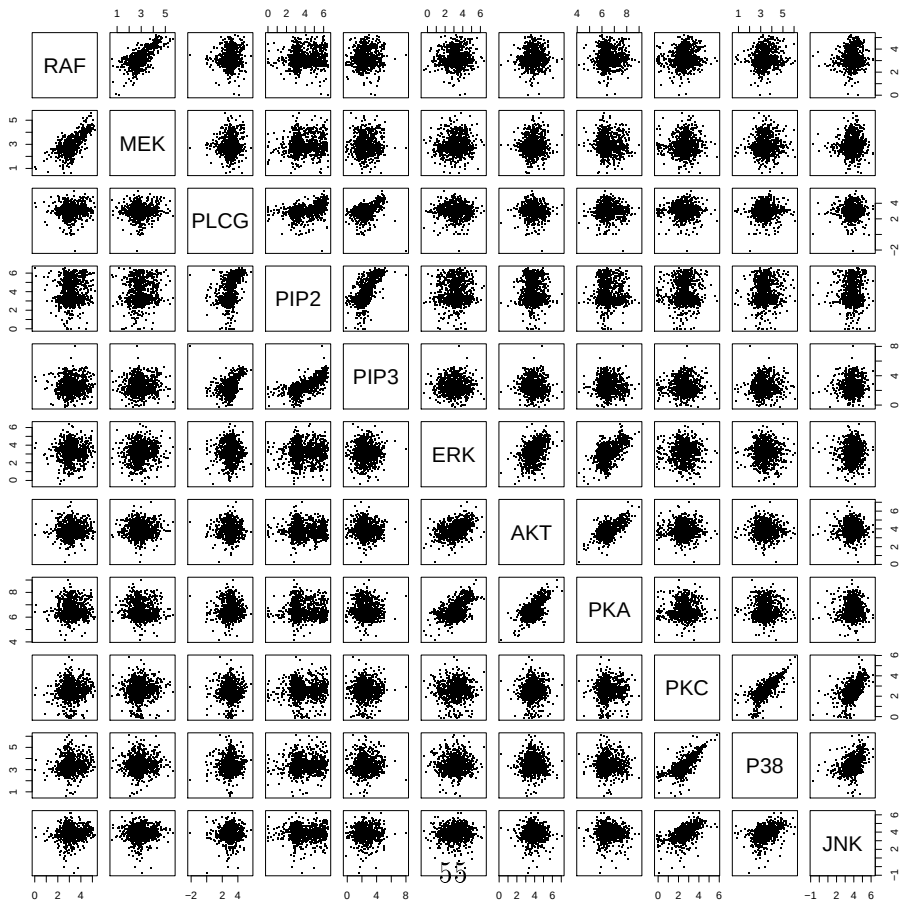


Figure 3.7: Plot of the Simulate dataset 8 - residuals/1.

3.3 Performances in terms of finding dependencies

3.3.1 Application to eight datasets

Grouping the 8 datasets

When applying the algorithms, we expect to reconstruct a skeleton as illustrated in Figure 3.8. We assume the ground truth to be formed by the "well known dependencies".

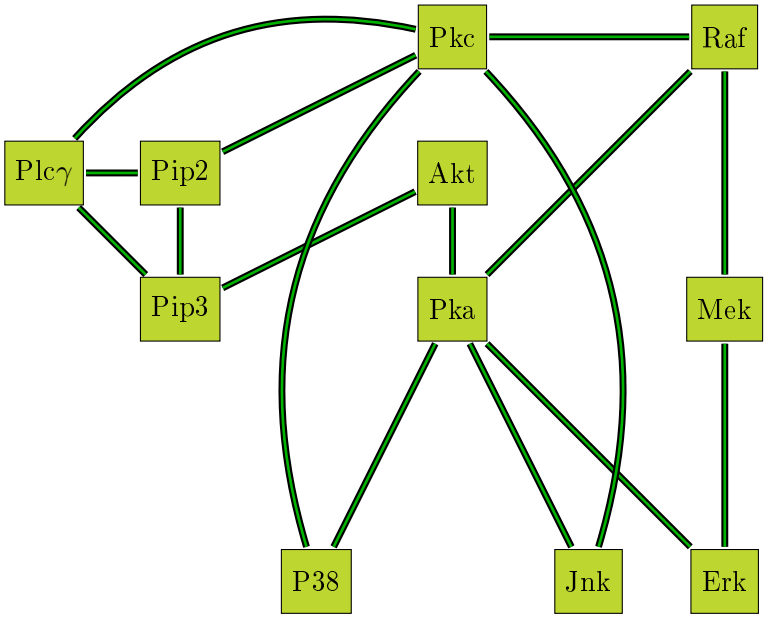


Figure 3.8: The skeleton to which the output of the algorithms is compared.

In order to obtain an output graph as in Figure 3.8, we use the information acquired from the 8 datasets. Only summarizing the results gained from each dataset we can end up finding the real underlying structure of the data. To recover the skeleton of the graphical structure, we developed three methods, which are now explained:

I. *All edges.*

II. *Second = 1.*

III. *Ceil Floor.*

The method I puts in the model all the edges that are found applying the discovery algorithm to every datasets. The methods II and III rely on the fact that in the first two datasets the measurements are performed after a

general perturbation, while in datasets 3 - 8 the perturbations are specific. Both methods count how many datasets include each edge in the model. We can display this summary as a graph where each edge is labeled with a number that represents in how many datasets that edge has been included. We refer to this graph as $\mathcal{M}_A$. We calculate the mean value of the number of appearances of the edges. For every 8 datasets and a specified algorithm we obtain the mean value $\hat{m}$. It is highly likely that this number is not an integer. We define $f = \lfloor \hat{m} \rfloor$ and $c = \lceil \hat{m} \rceil$. The methods II and III include in the final model all the edges of $\mathcal{M}_A$ labeled with a number higher or equal to c. We call this model $\mathcal{M}_1$. Considering now the datasets from 3 to 8, as before, we calculate how many times an edge is included in the 6 graphs. We then summarize the number of appearances of each edge with a graph $\mathcal{M}_B$, as done with $\mathcal{M}_A$. Method II adds to $\mathcal{M}_1$ the edges that appear at least once in $\mathcal{M}_B$. Method III puts in the final model the edges which appear at least $f - 1$ times in $\mathcal{M}_B$. Those two methods are very similar to each other, III is more restrictive than II when $f > 2$. The idea of these two ways of inspecting datasets is that first of all we need to include in the final model the edges which are very likely to play a central role in the causal relation. We suppose these dependencies to be the ones which are the most present in the 8 models. We then include in the model also dependencies which are found only by a specific intervention on a protein. This explains why we also add to the model edges which appear less often but help to elucidate causal relationships.

Real data

Table 3.2 illustrates the performances of the algorithms kPCs, bPC and PC. The free parameters have been fixed as follows:

- # of Permutations = 300 for kPC Permutation.
- # of Permutations = 500 for bPC.
- $\epsilon = 0.1$ for kPC Permutation.
- $h = 50$ for kPC Permutation.
- $\sigma = \{1, 5, 9\}$ for kPCs.

The output is analyzed at different α thresholds.

REAL DATA

		kPC Residuals			kPC Permutation			bPC	PC
		$\sigma = 1$	$\sigma = 5$	$\sigma = 9$	$\sigma = 1$	$\sigma = 5$	$\sigma = 9$		
$\alpha = 0.10$	All edges	8(9)	9(10)	11(13)	8(10)	9(11)	10(11)	9(10)	11(13)
		2(1)	3(2)	4(2)	3(1)	3(1)	3(2)	3(2)	6(4)
	Second = 1	8(9)	9(10)	11(12)	8(9)	9(10)	10(11)	9(10)	11(12)
		2(1)	2(1)	2(1)	2(1)	2(1)	2(1)	3(2)	5(4)
	Ceil Floor	6(7)	7(8)	8(9)	7(8)	8(9)	8(9)	8(9)	8(9)
		2(1)	2(1)	2(1)	2(1)	2(1)	2(1)	2(1)	2(1)
$\alpha = 0.15$	All edges	9(10)	11(13)	11(13)	8(10)	9(11)	10(12)	11(12)	12(14)
		2(1)	6(4)	7(5)	6(4)	7(5)	9(7)	3(2)	11(9)
	Second = 1	9(10)	11(12)	11(12)	8(9)	9(10)	10(11)	11(12)	12(14)
		2(1)	3(2)	4(3)	2(1)	4(3)	6(5)	2(1)	7(6)
	Ceil Floor	7(8)	8(9)	8(9)	8(9)	8(9)	8(9)	8(9)	9(10)
		2(1)	2(1)	2(1)	2(1)	2(1)	2(1)	2(1)	2(1)

Table 3.2: Comparison among different methods of grouping datasets at different thresholds α and with different algorithms. The analyzed data are the real ones after a log-transformation.

Fixed an algorithm, a threshold α and a method to group the 8 datasets, every box is made of four numbers. The top ones represent the number of correct edges found by the algorithm, the bottom ones the number of wrong connections. The output of each algorithm has been compared with Figure 3.8. The "well-known dependencies" are sixteen. The numbers in brackets represent the output if we take into account that the edges "known dependencies" and the ones that could arise from the fact we do not have antibodies specific to RAF S497 S499 are correct.
Among the wrong edges, MEK - AKT and P38 - JNK are always found. The former belongs to the list "known dependencies", the latter has been found also by other statisticians (Eaton and Murphy 2007; Hyttinen, Eberhardt, and Hoyer 2010). Looking at the two kPC algorithms, we see that augmenting σ and α always leads to finding more correct edges. While the increase of α also contributes to the detection of more wrong connections, the increase of σ is more stable under this point of view. We observe that the method Ceil Floor is very little affected by changing of σ and α. As regards the bPC, we remark that it has higher performances with higher α. In fact, when increasing α, only the number of correct edges increase.
Finally, comparing kPCs and bPC with PC, we notice that PC could find as many edges as the other three previous algorithms. This always comes with a discovery of more wrong edges. We can say without any doubt that on average, PC always finds more wrong edges, which makes this method more useless.

RESIDUALS / 1

		kPC Residuals $\sigma = 1$	kPC Residuals $\sigma = 5$	kPC Residuals $\sigma = 9$	kPC Permutation $\sigma = 1$	bPC	PC
$\alpha = 0.10$	All edges	11	12	10	12	13	11
		7	11	8	14	8	12
	Second = 1	11	11	9	11	12	9
		7	8	6	10	6	9
	Ceil Floor	8	8	8	10	8	8
		0	0	0	0	0	0
$\alpha = 0.15$	All edges	13	12	12	13	12	11
		11	13	10	13	11	14
	Second = 1	12	11	11	11	11	10
		9	10	7	9	8	10
	Ceil Floor	8	8	8	10	8	8
		2	1	1	2	1	3

Table 3.3: Comparison among different methods to group datasets at different threshold α and with different algorithms. The analyzed data are the simulated data - residuals/1.

Simulated data

As in Table 3.2, Tables 3.3-3.5 represent the number of correct and wrong connections with the free parameter fixed as previously. In this setting, we do not have numbers in brackets because we exactly know how the data have been generated. The way of simulating the data is the one described in Chapter 3.2 and we consider the ground truth as the one illustrated in Figure 3.8.

As before, the trend is that augmenting α makes the numbers of correct and wrong edges increase. With those data, we notice that the number of wrong edges is high comparing to Table 3.2. The Ceil Floor method works very well in removing lots of them. As the noise decreases, all the algorithms detect more correct and less wrong connections. Until fifteen edges up to sixteen edges are found. An edge which is always among the missing edges is PLCγ-PKC. We can guess that this is related to the fact that the simulations are not perfect, because we do not have antibodies specific to RAF S497 - S499. Otherwise, this could be because the algorithms are not good in detecting the type of dependence between those two variables. Another reason could be that PKC is much more influenced by PIP2 than by PLCγ. In fact, in every dataset but one the SNR of PIP2 is higher than the SNR of PLCγ (see Table 3.6). The Figure 3.9 represents PKC vs PLCγ in dataset 8 when the residuals are reduced by a factor of ten. We see that the dependence is very difficult to notice. Furthermore, after choosing a method among kPCs

RESIDUALS / 3

		kPC Residuals			kPC Permutation	bPC	PC
		$\sigma = 1$	$\sigma = 5$	$\sigma = 9$	$\sigma = 1$		
$\alpha = 0.10$	All edges	14	14	14	15	14	13
		5	6	7	4	3	8
	Second = 1	14	13	13	15	14	12
		5	6	7	4	2	7
	Ceil Floor	10	9	11	9	9	10
		0	1	1	0	0	2
$\alpha = 0.15$	All edges	14	14	14	15	14	13
		9	9	9	6	6	11
	Second = 1	14	13	13	15	14	12
		9	9	9	6	3	9
	Ceil Floor	11	12	10	12	11	12
		0	1	1	0	0	2

Table 3.4: Comparison among different methods to group datasets at different threshold α and with different algorithms. The analyzed data are the simulated data - residuals/3.

RESIDUALS / 10

		kPC Residuals			kPC Permutation	bPC	PC
		$\sigma = 1$	$\sigma = 5$	$\sigma = 9$	$\sigma = 1$		
$\alpha = 0.10$	All edges	13	15	15	15	14	15
		3	4	4	3	2	10
	Second = 1	13	15	15	15	14	14
		2	3	3	2	2	9
	Ceil Floor	11	10	10	10	12	11
		0	0	0	0	0	4
$\alpha = 0.15$	All edges	15	15	15	15	15	16
		5	7	7	5	4	11
	Second = 1	15	15	15	15	15	15
		3	5	5	2	3	10
	Ceil Floor	12	12	12	10	12	11
		0	0	0	0	0	5

Table 3.5: Comparison among different methods to group datasets at different threshold α and with different algorithms. The analyzed data are the simulated data - residuals/10.

and bPC, an α, the outputs of each algorithm are very similar to each other. Investigating the PC algorithm, we can observe that in residuals/1 and residuals/3 it finds causal structures with less correct and more wrong edges than kPCs and bPC. When the residuals are ten times smaller, it succeeds in finding the 16 edges, but the big inconvenient is that also 11 wrong edges are found. This, again, makes this algorithm less useful.

SNR in Simulated Data - Residuals/10

	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Data8
PLCγ	3.85	61.28	19.28	28.35	1.30	5.13	3.55	34.56
PIP2	27.84	125.36	33.52	0.14	17.38	61.00	25.02	NA

Table 3.6: SNR of the PLCγ and PIP2 in the 8 datasets with data: simulated data - residuals/10.

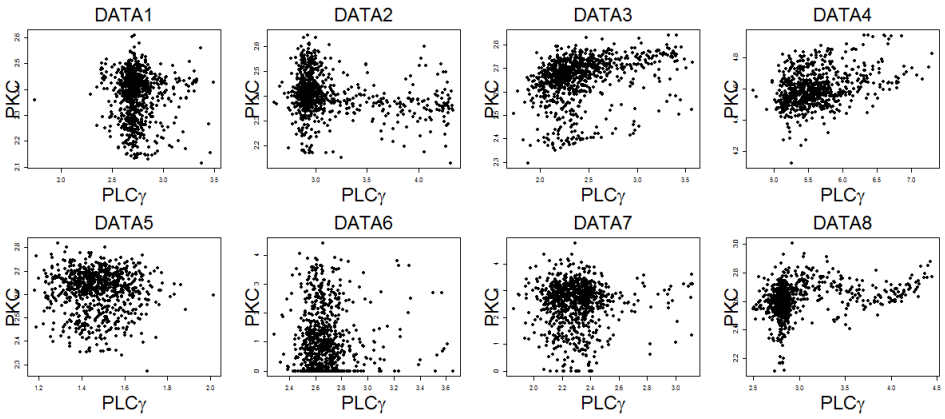


Figure 3.9: PLCγ vs PKC in simulated dataset - residuals/10.

Conclusion

Comparing real and simulated data, as we observed previously, their plots and boxplots are very similar. Despite these similarities, the outputs of the algorithms are quite different. When we compare data with the same amount of noise, the performances in terms of correct edges are always better with simulated data - residuals/1 than with real ones. This could be due to the fact that when we generate the data, they are more structured and ordered compared to the real ones. Hence, it is simpler to detect dependencies.
Furthermore, we also discovered that with the data generation more wrong edges are found. This is because each dataset creates a graph which has

different wrong edges than other datasets. When in *All Edges* all found connections become part of the final model, more wrong ones are included. This could be due to the fact that with raw data, the datasets are "dependent" because the measurements have been taken in the same laboratory, with the same tools, in the same moment. With simulated data, each dataset arises as completely independent from any other.
Comparing the three different ways of grouping datasets, we can affirm that *All Edges* is an unsuitable method, because including all edges in the final model, it also includes lots of wrong ones. There is no "pre-selection" of connections. The method *Second=1* leads to more edges than *Ceil Floor*: the selection is not strict enough. In fact, *Second = 1* works better with the real data where the number of wrong edges is low comparing to the simulated data. With the simulated data the *Ceil Floor* method has higher performances since more edges which appear few times in the eight models are forbidden to enter in the final model.
Finally, we can assert that the PC is never as highly performing as kPCs and bPC. This is due to the fact that its final causal model lacks of dependencies or has plenty of wrong edges or both.

3.3.2 Application to one dataset

In order to compare the performances of the algorithms, we will use the ROC curve. The used data are the simulated - residuals/1 from dataset 8. The free parameters are fixed as in Table 3.2 and for kPC Permut - Resid and kPC Residuals and σ=1. The output of the algorithms is compared with the graph in Figure 3.5
Looking at Figure 3.10, we see that PC is always less efficient than the other discovery algorithms. In this particular setting, kPC Permut - Resid is the best one. The algorithms kPCs and bPC always have an area under the curve greater that 0.8, which outlines the good performances of these methods.
Figure 3.11 shows the different ROC curves of kPC Residuals when σ varies. We can conclude that there are not many differences. Moreover, the performances are the same when $\sigma = 5$ and $\sigma = 9$.

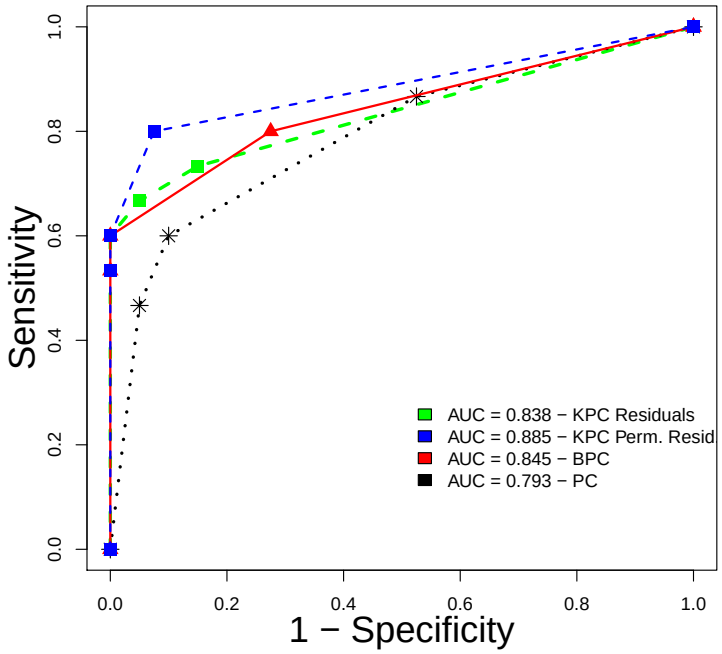


Figure 3.10: ROC curve to compare kPCs, bPC and PC algorithms. The data are the simulated ones - residuals/1 from dataset 8.

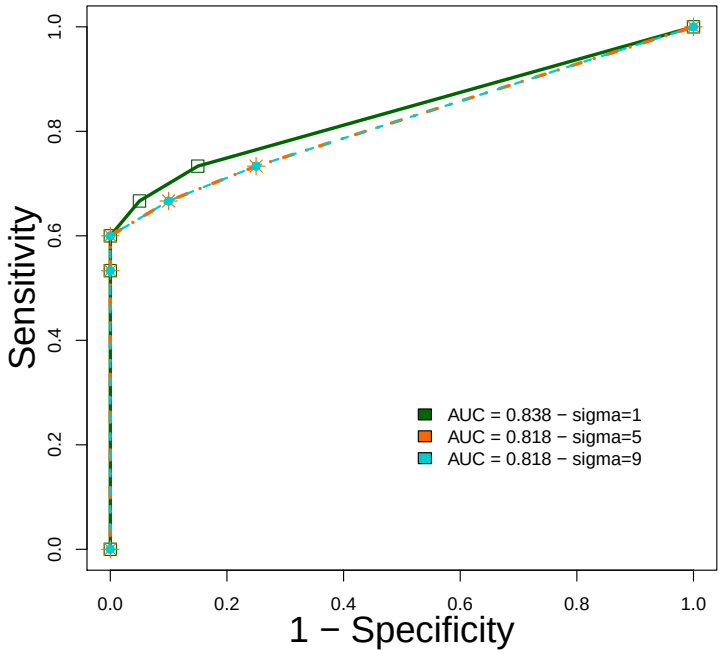


Figure 3.11: ROC curve with simulated data - residuals/1. The used algorithm is the kPC Residual with different kernel widths σ.

3.4 Performances in terms of discovering the direction of the causal orientations

We are now interested in finding as much oriented edges as possible thanks to the Generalized transitive step. For sake of simplicity, we will focus our attention on one dataset only: dataset 8. The edges are orientated first in the Collider step and then in the Transitive step. When the Generalized transitive step will be used, in order to differentiate it from the algorithm with the Transitive step, we will designate the algorithm adding to his name '$\mathfrak{C}$omplete', shortened as $\mathfrak{C}$. See Table 3.7.
Table 3.8 illustrates how many oriented edges are found. After having chosen an algorithm, the kind of data to use and one of the two steps, every box is made of two numbers. The top one represents the number of correct orientations found, the bottom one the number of wrong orientations. The last row describes the total number of found orientations. The parameters which have to be fixed are the same used to do Table 3.2.
We remark that the $\mathfrak{C}$omplete version of the algorithms always finds more causal directions.
Using the Generalized transitive step, it is possible to successfully orient all the edges of the output graph when the Transitive step fails.
Figures 3.12 - 3.14 illustrate the output graph at the end of the three steps when using the algorithm kPC Residuals $\mathfrak{C}$ with simulated data - residual/1, while Figure 3.15 shows which would have been the output using the Transitive step. There is a huge difference between the two final outputs. The one with the Generalized transitive step has four more directed edges and all the edges have been orientated: all of them but one are one-side arrowed. The double arrowed edge is AKT $\leftrightarrow$ PIP3, which has been orientated in this way in the Collider step. Moreover, there is only one edge for which the causal orientation is in the wrong direction.
It might happen that there is not enough information to infer the orientations in the Collider step. This would lead not to be able to detect any direct causality in the Transitive step. Analyzing the residuals with the Generalized transitive step we succeed in orienting edges. Figures 3.17 - 3.20 illustrate the output graph at the end of the three steps when using the algorithms kPC Permutation and kPC Permutation $\mathfrak{C}$ with real data . The probabilistic

PHASE	Algorithm **x**	Algorithm **x**-$\mathfrak{C}$
I	Skeleton step	Skeleton step
II	Collider step	Collider step
III	Transitive step	Generalized transitive step

Table 3.7: Comparison between algorithm "standard"(left) and "$\mathfrak{C}$omplete" (right).

REAL DATA

	kPC Resid	kPC Resid - $\mathfrak{C}$	kPC Perm	kPC Perm - $\mathfrak{C}$	bPC	bPC - $\mathfrak{C}$
STEP II	0	0	0	0	0	0
	0	0	0	0	0	0
STEP III	0	3	0	3	0	2
	0	1	0	1	0	1
TOT	0	3	0	3	0	2
	0	1	0	1	0	1

SIMULATED DATA - RESIDUALS/1

	kPC Resid	kPC Resid - $\mathfrak{C}$	kPC Perm	kPC Perm - $\mathfrak{C}$	bPC	bPC - $\mathfrak{C}$
STEP II	2	2	2	2	4	4
	1	1	1	1	1	1
STEP III	0	4	0	4	0	2
	0	0	0	0	0	0
TOT	2	6	2	6	4	6
	1	1	1	1	1	1

Table 3.8: These table describe how many correct (top line) and wrong (bottom line) oriented edges are found during the different steps of the algorithms.

model is made of 3 three-variables structures where each of the three nodes is connected to each other plus by a two-variables structure. Because of this setting, it is not possible to orient any edges, neither in the Collider step nor in the Transitive step. With the Generalized transitive step we can differentiate among the oriented structures. The algorithm has worked as follow. First, the orientation RAF $\rightarrow$ MEK has been found. Secondly, analyzing AKT and its connected nodes PKA and ERK, the orientation PKA $\rightarrow$ AKT has been discovered. This caused the immediate orientation of AKT$\rightarrow$ ERK. Thanks to the Transitive step inside of the Generalized transitive step, PKA $\rightarrow$ ERK has been found. For sake of completeness Figures 3.22-3.25 illustrate the output of the bPC and bPC-$\mathfrak{C}$ with simulated - residuals/1 data. It is straightforward to notice that here too all the edges that were not oriented in Collider step have been oriented in the Generalized transitive step.
We can assert that the non-linearity of the data is very useful in finding the direction of the causal dependencies. This step works with great results not only with the simulated dataset but also with the real data.

Skeleton phase

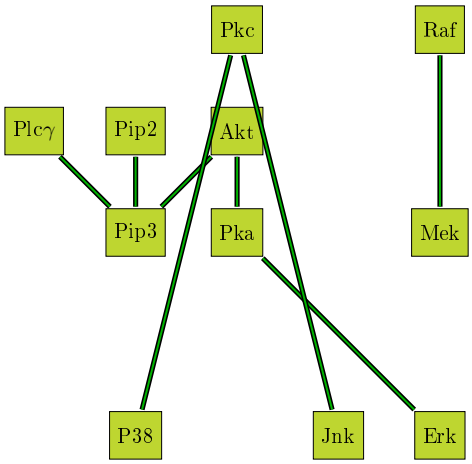


Figure 3.12

Collider phase

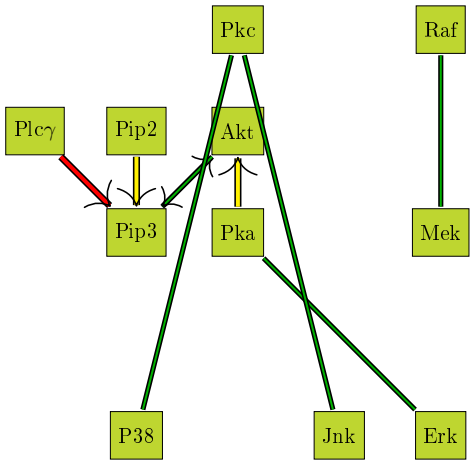


Figure 3.13

Generalized transitive phase

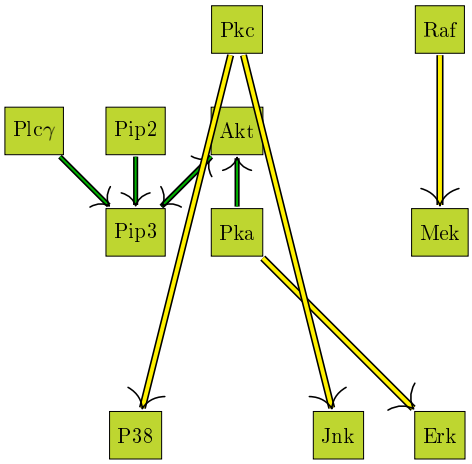


Figure 3.14

Transitive phase

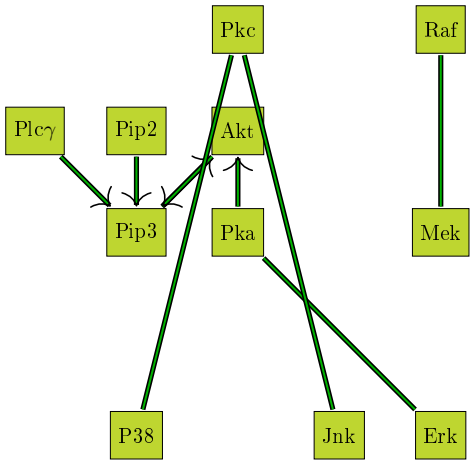


Figure 3.15

Figure 3.16: The four graphs illustrate the output of the phases of kPC Residuals and kPC Residuals - $\mathfrak{C}$ with simulated - residuals/1 data. Green undirected edges represent correct found edges. Yellow orientated edges represent correct orientations and red directed edges represent wrong - direction orientations. Green double directed edges represent correct edges for which we cannot infer any causal direction. For each phase the green arrowed edges are the orientations found in the previous step while the non-green edges are the orientations discovered in that phase.

Skeleton phase

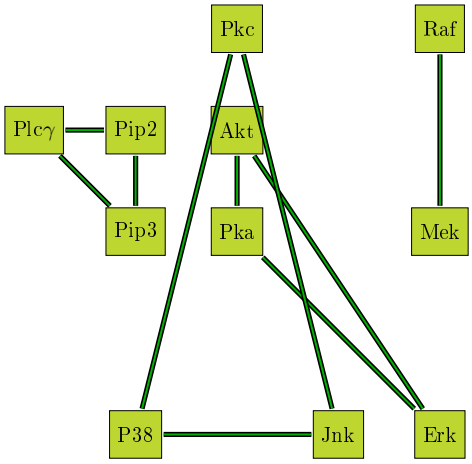


Figure 3.17

Collider phase

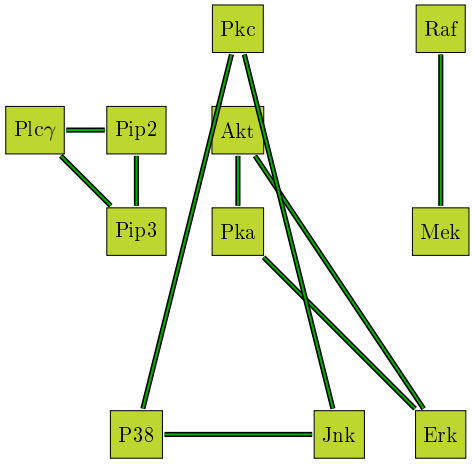


Figure 3.18

Generalized transitive phase

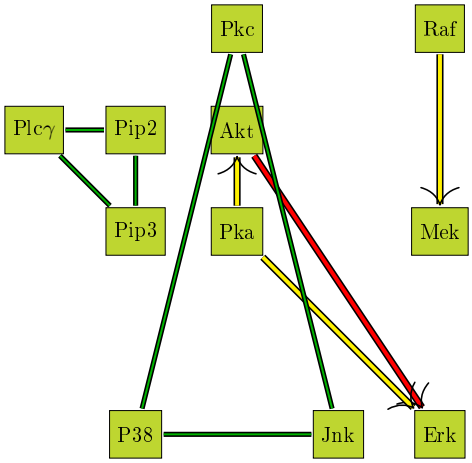


Figure 3.19

Transitive phase

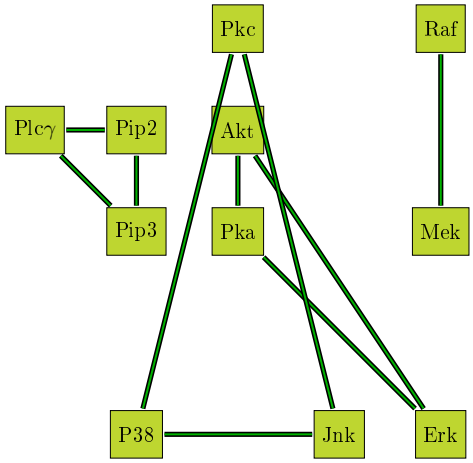


Figure 3.20

Figure 3.21: The four graphs illustrate the output of the phases of kPC Permutation - Residuals and kPC Permutation - Residuals - $\mathfrak{C}$ with real data. Green undirected edges represent correct found edges. Yellow orientated edges represent correct orientations and red directed edges represent wrong - direction orientations. In every phase the non-green edges are the orientations discovered in that phase.

Skeleton phase

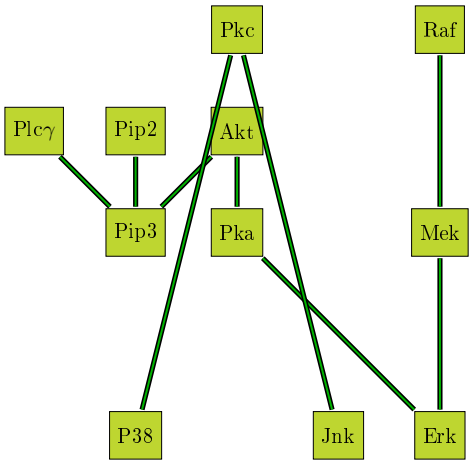


Figure 3.22

Collider phase

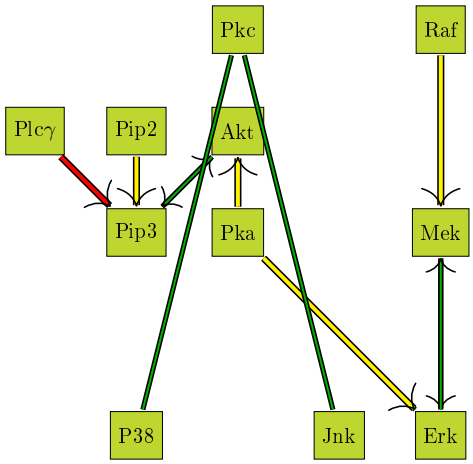


Figure 3.23

Generalized transitive phase

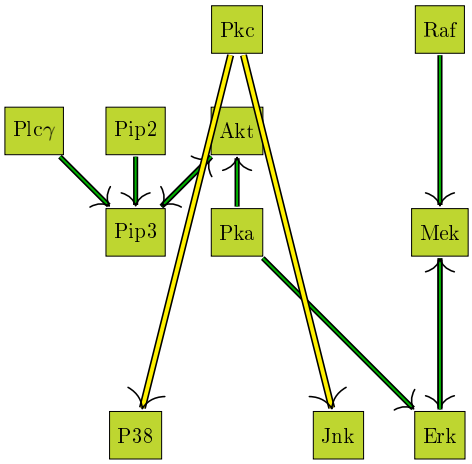


Figure 3.24

Transitive phase

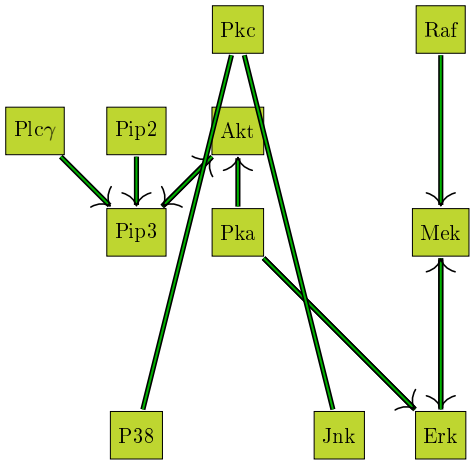


Figure 3.25

Figure 3.26: The four graphs illustrate the output of the phases of bPC and bPC - $\mathfrak{C}$ with simulated - residuals/1 data. Green undirected edges represent correct found edges. Yellow orientated edges represent correct orientations and red directed edges represent wrong - direction orientations. Green double directed edges represent correct edges for which we cannot infer any causal direction. For each phase the green arrowed edges are the orientations found in the previous step while the non-green edges are the orientations discovered in that phase.

4 SUMMARY AND FUTURE WORK

After investigating the methods kPC Residuals $\mathfrak{C}$, kPC Permutation $\mathfrak{C}$ and bPC $\mathfrak{C}$, we can assert that they are valuable tools to discover causal relationships among non-linear and non-Gaussian data. Their usefulness lies in the fact that they all characterize independence and extend the way of finding the directions of the causality.
Both kPCs algorithms rely on the characteristic kernel we choose to use. The Gaussian kernel seems to be a very flexible tools since it enables us to find independence in many different data settings. The kPC already existed, but only in the version kPC Permutation - Cluster $\mathfrak{C}$ (Tillman, Gretton, and Spirtes 2009).
The bPC $\mathfrak{C}$ algorithm has been implemented and tested for the first time here combining the PC and the Brownian distance correlation. First of all, we notice that the distance correlation is a natural extension of the Pearson correlation parameter (Székely, Rizzo, et al. 2009). Assuming we are dealing with two random variables X and Y, it is highly likely that X and Y have been generated by the random process of Brownian motion. Thanks to the results in Chapter 2.4, we can affirm that measuring the distance correlation is the same as measuring the covariance of random vectors with respect to the Wiener process.
The kPCs are influenced by the kernel width σ, which plays an important role in determining the nature of the relation among variables. It is a difficult parameter to set, and it is closely related to the distance there is from one observation to the others. As in kPCs, the bPC too is affected by a parameter: $\xi \in (0, 2)$, which characterizes the type of distance we are interested in when calculating $\nu^2(X, Y; w)$. All the three analyzed algorithms have always given very similar output. We cannot find any strong evidence against or in favor of one of them.
Regarding the computational efficiency, bPC and kPC Residuals are much quicker than kPC Permutation Residuals. This is due to the fact that in this latter method, the calculation of the HSIC is quite expensive and must

be computed p times. The bPC, too, involves the number of permutations p, but calculating the distance correlation is more rapid than measuring the HSIC. As we can see in Figure 4.1, kPC Permut-Residuals can take up to more than one hundred times longer to discover a DAG. We solved this problem parallelizing the algorithm during the permutation test. Assuming we have p values of HSIC to estimate and k CPU, we make every CPU calculate $\frac{p}{k}$ values of HSIC. This won't reduce the computational time by a factor k since we parallelized only part of the code, but it helps in speeding the algorithm. Looking at Figure 4.2 we see that there are differences also among kPC Residuals and bPC. The latter is faster then the former. As said before, it is because the distance covariance is computationally easier to measure.
As we investigated in Chapter 3, the algorithms might not find all dependencies. It could be useful to expand those algorithms including the detection of latent variables. It is in fact very common not to be able to measure all the variables which should enter in the model as we saw in the previous chapter. Some dependencies could be missing because those algorithms are structured in order to give a DAG as an output. It is in fact likely that biological networks include cycle. It would be interesting to extend our algorithms so that they can discover cyclic pathways.
Finally, it could be compelling to inspect the effects of the manipulation: to use the found dependencies to generate more knowledge about the functioning of a cell, which leads to learn more about a disease and have more tools to treat it.

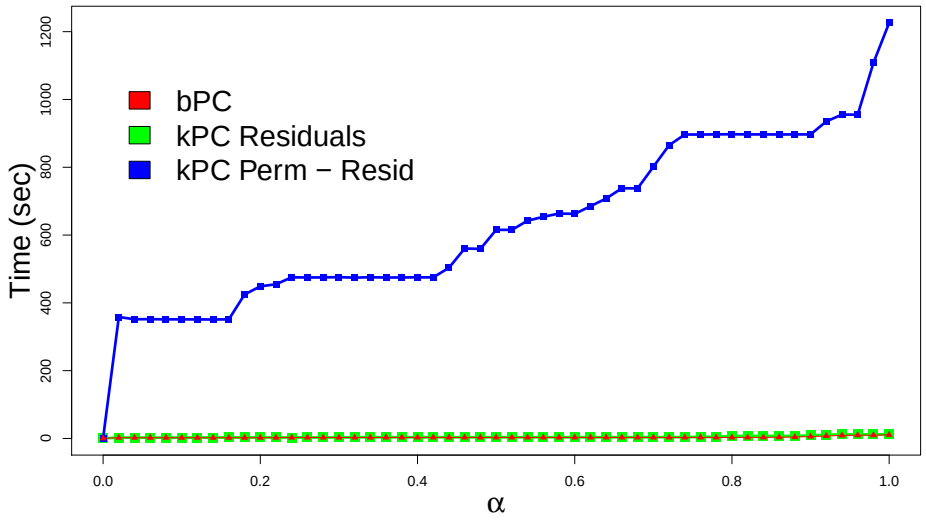


Figure 4.1: Time it takes to the algorithms to find the causal models with different thresholds α. The graph that should be discovered is the one in Figure 2.62 which is made by five variables and five edges.

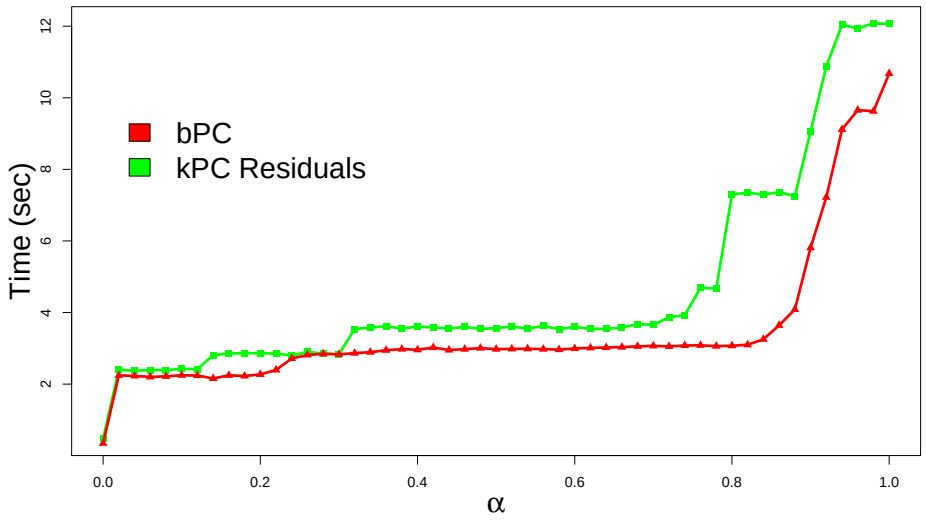


Figure 4.2: Zoom of Figure 4.1 to catch the differences among kPC residuals and bPC.

5 APPENDIX

Here are reported the main function `bPC()` and the most important auxiliary functions

- `Skeleton()`
- `Collider()`
- `GenTransitive()`

of the implemented algorithms. For sake of simplicity we only show the bPC algorithm. The only differences between the bPC and the kPCs implementation lay in the independent and conditional indpendent tests implementation in the `Skeleton` and `GenTransitive` functions:

- condIndepTestBr(x= data[,i], y= data[,ad[j]], z = data[, sub[1:N , k]], p = p) in `Skeleton()`
- IndepTestBr(x = data[,i], y = data[, ad[j]], p = p) in `Skeleton()`
- indepResid_bpc(data = data, G = G, undiEdge = undiEdge, v = nodeS[i], parent = parent, S = sub[,j], alpha = alpha, p = p) in `GenTransitive()`

```
bPC<- function(data,  # the available data
               alpha, # level of the test
               p)     # number of permutation in dcov.test

  # it loads all the auxiliary functions needed
  source('Source_bPC.R')
  Source_bPC()

  # G       Adjacent matrix with all the undirected edges
  # G_dir   Adjacent matrix with ONLY the directed edges
  # G_ad    Adjacent matrix directed and undirected edges
  # Adj_Mat Final adjacent matrix
```

```
  ## SKELETON step

  Sk      = Skeleton(data, alpha, p)
  sepset  = Sk$sepset
  G       = Sk$G

  ## COLLIDER step

  Co = Collider(G, sepset)
  G_dir = Co$G_dir
  G_ad  = Co$G_ad
  Coll  = Co$collider

  ## GENERALIZED TRANSITIVE step

  undiEdge = updateUE(G_ad, NULL, NULL)
  s = 1

  Adj_Mat  = GenTransitive(data, G_ad, G_dir, undiEdge, s, alpha, p)

  list(mat = Adjacency_Matrix, Coll = Coll, Gskel = G, Gcoll = G_ad)


#################################################################

Skeleton <- function(data,   # the available data
                     alpha,  # level of the test
                     p){     # num of permutation in dcov.test

  n = ncol(data)
  G = diag(x = -1, nrow = n, ncol = n)+1 # graph matrix with 0 on the diagonal
                                         # (complete graph)
  sepset = list()  # I create the separation set
  for(i in 1:n)
  {
    sepset[[i]]  = list()
  }
  check = hash() # I create an auxiliary tool to keep in memory
                 # which tests have already been performed
  for (N in 0:n) # for each N = dimension subset to be taken
  {
    if(N!=0)
    {
```

```
      G_old = G
        for (i in 1:n) # for each node i
        {
          ad = adiac_D(G_old, i) # the adjacent node to vertex i (undirected edge)
          n_ad = length(ad)
          if((n_ad-1)>=N) # if the dimension of ad >= N
          {
            for ( j in 1:n_ad) # for each node ad[j] adjacent to i
            {
              if(n_ad==2) # I consider all the subsets of ad\{ad[j]}
                          # of dimension N
              {
                sub = c(ad[3-j]) # because combn(8, 1)
                                 # returns 1, 2, 3, 4, 5, 6, 7, 8
                sub = as.matrix(sub)
              }
              else # I consider all the subsets of ad\{j} of dimension N
              {
                sub = combn(c(ad[-c(which(ad==ad[j]))]), N)
              }
              k = 1
              while(k<=ncol(sub) && G[i, ad[j]]==1 ) #while there are subsets
                 # to investigate for c.i. AND I still haven't found a subset
                 # for which i and adj[j] are c.i.
              {
                a = c(i, ad[j])       # procedure to keep in memory
                                      # which c.i. have been tested
                a = sort(a)
                a = paste(a, collapse="")
                b = sort(sub[1:N, k])
                b = paste(b, collapse="")
                long = paste(a, b, sep='x')
                if( is.numeric(check[[long]]) == F ) # if I still haven't tested
                                                     # the dep of i and ad[j]
                                                     # given sub[1:N , k]
                {
# TEST OF CONDITIONAL DEPENDENCE: H_0: i is indep to ad[j] given sub[1:N , k]
#                                 H_1: i and ad[j] are dependent given sub[1:N , k]
dep = condIndepTestBr(x= data[,i], y= data[,ad[j]],
                      z = data[, sub[1:N , k] ], p = p)
                  if (dep > alpha) # if the pval > alpha,
                                   # I DON'T refuse H0
                                   # so i and adj[j] are c.i.
                  {
```

```
                    G[i, ad[j]] = 0 # I delete the connection betwwen i and ad[j]
                    G[ad[j], i] = 0
                    # I record the separation set in sepset
                    sepset[[i]][[ad[j]]] = sub[1:N,k]
                    sepset[[ad[j]]][[i]] = sub[1:N,k]
                  }
                k = k+1
                check[[long]]=1
                } # end if
              } # end while
            } # end for ( j in 1:n_ad)
          } # end if((n_ad-1)>=N)
        } # end for (i in 1:n)
    } # end if (N!=0)
    else # when N==0
    {
      for ( i in 1:n)
      {
        ad = adiac_D(G, i) #  the adjacent node to vertex i (undirected edge)
        n_ad = length(ad)
        if((n_ad-1)>=N) # if the dimension of ad >= N
        {
          for ( j in 1:n_ad) # for each node ad[j] adjacent to i
          {
            # TEST OF DEPENDENCE: H_0: i and ad[j] are independent
            #                     H_1: i and ad[j] are dependent
            dep = IndepTestBr(x = data[,i], y = data[, ad[j]], p = p)
            if (dep > alpha) # if the pval > alpha,
                             # I DON'T refuse H0
                             # so i and adj[j] are indep.
            {
              G[i, ad[j]] = 0 # I delete the connection betwwen i and ad[j]
              G[ad[j], i] = 0
              sepset[[i]][[ad[j]]] = -1 # it's a  default value
              sepset[[ad[j]]][[i]] = -1
            }
          }
        }
      }
    }
  }
  Sk = list(sepset = sepset, G = G)
  Sk
}
```

```
#################################################################

Collider <- function(G,        # skeleton matrix
                     sepstep){ # separation set
  n = ncol(G)
  G_ad = matrix(data = 0, nrow = n, ncol = n)
  coll = list()
  l=1
  for(i in 1:n) # for each node i
  {
    ad_i = adiac_D(G, i) # nodea adjacent to i
    if( length(ad_i) > 1 ) # if node i has more than 1 neib
    {
      for(j in 1:(length(ad_i)-1)) # I take every node ad[j] adjacent to i
      {
        for(k in (j+1):length(ad_i))# and another one
        {
          # if ad_i[j] and ad_i[k] are not adjacent
          # and i is not in sepstep[[ ad_i[j] ]][[ ad_i[k] ]]
      if (G[ad_i[j], ad_i[k]]==0 && !(i\%in\%sepstep[[ ad_i[j] ]][[ ad_i[k] ]]))
          {
            G_ad[ad_i[j], i] = 2 # I create the collider NUMBER 2
            G_ad[ad_i[k], i] = 2
            coll[[l]] = c(ad_i[j], i, ad_i[k])
            l = l+1
          }
        }
      }
    }
  }
  G_dir = G_ad # in G_dir there are only the directed edges
  for(i in 1:(n-1)) # I construct the matrix G_ad with also the undirected edges
  {
    for(j in (i+1):n)
    {
      if(G[i, j]==1 && G_ad[i, j]==0 && G_ad[j, i]==0) # unuseful G[j, i]==1
      {
        G_ad[i, j] = 1
        G_ad[j, i] = 1
      }
    }
  }
  Co = list(G_dir = G_dir, G_ad = G_ad, collider = coll )
```

```
}

#################################################################

GenTransitive <- function(data,        # available data
                          G_ad,        # adjacency matrix with all edges
                          G_dir,       # adjacency matrix with only directed edges
                          undiEdge,    # auxiliary tool
                          s,
                          alpha,       # level of the test
                          p            # number of permutations
){
  G = G_ad
  n = ncol(G)
  numUndiEdge = undiEdge[[n+1]] # it keeps in memory the number
                                # of undirected edges linked to each node
  updated     = undiEdge[[n+2]] # it keeps in memory in nodes updated
                                # in the previous step
  maxUndiEdge = max(numUndiEdge)
  while(maxUndiEdge >= s)
  {
    nodeSa = which(numUndiEdge == s) # I consider only nodes that have
                                     # s undirected edge
    # and nodes that have less than s undirected edge
    # and have been updated in the previous step
    nodeSb = which(numUndiEdge < s & numUndiEdge > 0 & updated ==1  )
    nodeS = c(nodeSa, nodeSb)
    if(length(nodeS)>0)
    {
      for ( i in 1:length(nodeS)) # for each nodes
      {
        undiEdge[[n+2]][nodeS[i]] = 0
        vectUndEdgeUi = undiEdge[[ nodeS[i] ]] #vector containing the nodes
                                               # to which nodeS[i] is related
                                               # with undir edge
        parent = adiac_S_Enter(G, nodeS[i])  # parents of nodeS[i]
        t = s
        while ( t > 0 )
        {
          if( t <= length(vectUndEdgeUi) )
          {
            if(length(vectUndEdgeUi) == 1)
            {
              sub = vectUndEdgeUi
```

```
            sub = t(as.matrix(sub)) # subset of the undir node
                                    # of cardinality t
          }
          else
          {
            sub = combn(vectUndEdgeUi, t) # subset of the undir node
                                          # of cardinality t
          }
          j = 1
          while ( j <= ncol(sub) )
          {
            # if linking nodeS[i]-sub[,j] does not create immorality
            if( NOTImmor(G, nodeS[i], sub[,j]) == 1)
            {
ir = indepResid_bpc(data = data, G = G, undiEdge = undiEdge, v = nodeS[i],
                    parent = parent, S = sub[,j], alpha =  alpha, p = p)
              if( ir == 1) # if all the independece assumption are fulfilled
              {
    # I create the orientation induced by the found dependence relationships
                compl = vectUndEdgeUi[ -c(which(vectUndEdgeUi == sub[,j]))  ]
                G[sub[,j], nodeS[i]] = 1
                G[nodeS[i], sub[,j]] = 0
                G_dir[sub[,j], nodeS[i]] = 1
                G_dir[nodeS[i], sub[,j]] = 0
                if(length(compl)>0)
                {
                  G[compl, nodeS[i]] = 0
                  G[nodeS[i], compl] = 1
                  G_dir[compl, nodeS[i]] = 0
                  G_dir[nodeS[i], compl] = 1
                }
                aux = G_dir
                FD = Final_Direction(G, G_dir) # I apply the transitive step
                G     = FD$G_ad
                G_dir = FD$G_dir
                diffFD = which(aux!=G_dir)
                if(length(diffFD) > 0) # and update the final graph
                {
                  upd = rep(0,n)
                  for( k in 1:length(diffFD))
                  {
                    upd[ floor( (diffFD[k]-1)/n) + 1 ] = 1
                    upd[ (diffFD[k]-1)\%\%n +1 ] = 1
                  }
```

```
                    upd[vectUndEdgeUi] = 1
                    upd = which(upd==1)
                  }
                  else
                  {
                    upd = vectUndEdgeUi
                  }
                  undiEdge = updateUE(G, c(nodeS[i], upd), undiEdge[[n+2]] )
                  t = 1
K = GenTransitive(data = data, G_ad = G, G_dir = G_dir, undiEdge = undiEdge,
                  s, alpha = alpha, p = p)
                  return(K)
                } # end  if( ir = 1)
              } # end if( NOTImmor(G, nodeS[i], sub[,j]) == 1)
              j = j+1
            } # end while ( j <= ncol(sub) )
          }
          t = t-1
        } # end while ( t > 0 )
      } # end for( i in 1:length(nodeS) )
    } # end if(length(nodeS)>0)
    s = s+1
  }# end while(maxUndiEdge >= s)
  G
}
```

List of Figures

List of Tables

List of Algorithms

Bibliography

Eaton, Daniel and Kevin P Murphy (2007). "Exact Bayesian structure learning from uncertain interventions". In: *International Conference on Artificial Intelligence and Statistics*, pp. 107–114.

Flesch, Ildikó and Peter JF Lucas (2007). "Markov equivalence in Bayesian networks". In: *Advances in Probabilistic Graphical Models*. Springer, pp. 3–38.

Fukumizu, Kenji et al. (2007). "Kernel Measures of Conditional Dependence." In: *NIPS*. Vol. 20, pp. 489–496.

Gretton, Arthur (2014). "Notes on mean embeddings and covariance operators". In: *Lectures*.

Gretton, Arthur et al. (2005). "Measuring statistical dependence with Hilbert-Schmidt norms". In: *Algorithmic learning theory*. Springer, pp. 63–77.

Gretton, Arthur et al. (2008). "A kernel statistical test of independence". In: *NIPS20*.

Hastie, Trevor and Robert Tibshirani (1986). "Generalized additive models". In: *Statistical science*, pp. 297–310.

Hauser Alain Buehlmann, Peter (2012). "Characterization and greedy learning of interventional Markov equivalence classes of directed acyclic graphs." In: *Journal of Machine Learning Research* 13, pp. 2409–2464. URL: `http://jmlr.org/papers/v13/hauser12a.html.`.

Hoyer, Patrik O et al. (2009). "Nonlinear causal discovery with additive noise models". In: *Advances in neural information processing systems*, pp. 689–696.

Hyttinen, Antti, Frederick Eberhardt, and Patrik O Hoyer (2010). "Causal discovery for linear cyclic models with latent variables". In: *on Probabilistic Graphical Models*, p. 153.

Kalisch Markus Maechler, Martin Colombo Diego Maathuis Marloes H. Buehlmann Peter (2012). "Causal Inference Using Graphical Models with the R Package pcalg." In: *Journal of Statistical Software* 47.11, pp. 1–26. URL: `http://www.jstatsoft.org/v47/i11/`.

Lauritzen, Steffen L (1996). *Graphical models*. Oxford University Press.

Mörters, Peter and Yuval Peres (2010). *Brownian motion*. Vol. 30. Cambridge University Press.

Pearl, Judea. *Probabilistic reasoning in intelligent systems: networks of plausible inference*.

— (2000). *Causality: models, reasoning and inference*. Vol. 29. Cambridge Univ Press.

Pe'er, Dana et al. (2001). "Inferring subnetworks from perturbed expression profiles". In: *Bioinformatics* 17.suppl 1, S215–S224.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: `http://www.R-project.org/`.

Rizzo, Maria L. and Gabor J. Szekely (2014). *E-statistics*. URL: `http://cran.r-project.org/web/packages/energy/energy.pdf`.

Sachs, Karen et al. (2005). "Causal protein-signaling networks derived from multiparameter single-cell data". In: *Science* 308.5721, pp. 523–529.

Shawe-Taylor, John and Nello Cristianini (2004). *Kernel methods for pattern analysis*. Cambridge university press.

Spirtes, Peter, Clark N Glymour, and Richard Scheines (2000). *Causation, prediction, and search*. Vol. 81. MIT press.

Székely, Gábor J, Maria L Rizzo, Nail K Bakirov, et al. (2007). "Measuring and testing dependence by correlation of distances". In: *The Annals of Statistics* 35.6, pp. 2769–2794.

Székely, Gábor J, Maria L Rizzo, et al. (2009). "Brownian distance covariance". In: *The annals of applied statistics* 3.4, pp. 1236–1265.

Tillman, Robert E (2009). "Learning Directed Graphical Models from Nonlinear and Non-Gaussian Data". In: *Master thesis*.

Tillman, Robert E., Arthur Gretton, and Peter Spirtes (2009). "Nonlinear directed acyclic structure learning with weakly additive noise model". In: *NIPS 22, Vancouver*.

Voortman, Mark and Marek J. Druzdel (2008). "Intensitivity of constraint-based causal discovery algorithms to violations of the assumption of multivariate normality". In: *FLAIRS*.

Wood, S.N. (2004). "Stable and efficient multiple smoothing parameter estimation for generalized additive models." In: *Journal of the American Statistical Association*. 99, pp. 673–686.

— (2011). "Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models." In: *Journal of the Royal Statistical Society*. 73.1.

Wright, Sewall (1918). "On the nature of size factors". In: *Genetics* 3.4, p. 367.

— (1921). "Correlation and causation". In: *Journal of Agricultural Resarch* 20.7.

— (1934). "The method of path coefficients". In: *The Annals of Mathematical Statistics* 5.3, pp. 161–215.

Acknowledgment

I wish to thank the MRC Biostatistics Unit of Cambridge for accepting me as a visiting student. In particular, I thank Sylvia Richardson and Lorenz Wernisch who supported me during those six months and taught me how to develop my project. This thesis could not have been accomplished without them.