Coupling aGrUM/pyAgrum with external libraries: an application to Copula Bayesian Networks

Marvin LASSERRE

supervised by Pierre-Henri WUILLEMIN (LIP6) and Régis LEBRUN (Airbus CRT)
directed by Christophe GONZALES (LIS)

March 18, 2022
• **Goal**: learning high dimensional continuous distributions with non-parametric models,
• **Goal**: learning high dimensional continuous distributions with non-parametric models,

• **Why high-dimensional?** Because complex systems involve a large number of variables,
• **Goal**: learning high dimensional continuous distributions with non-parametric models,

• **Why high-dimensional?** Because complex systems involve a large number of variables,

• **Why continuous?** Because in applications such as physics, engineering or finance variables are often continuous,
Learning high-dimensional probabilistic continuous models

- **Goal**: learning high dimensional continuous distributions with non-parametric models,
- **Why high-dimensional?** Because complex systems involve a large number of variables,
- **Why continuous?** Because in applications such as physics, engineering or finance variables are often continuous,
- **Why distributions?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),
- **Challenge 1**: Various non-parametric models exist to estimate a density but they are limited to a few dimensions (∼5 variables),
- **Solution**: Use of Probabilistic Graphical Models (PGM) to break the joint distribution into a product of conditional distributions of lesser dimensions.
- **Challenge 2**: We want a probabilistic model with a density from which we can sample points but continuous PGM are not satisfying,
- **Solution**: Use of the Empirical Bernstein Copula to parameterize graphical models.
• **Goal**: learning high dimensional continuous distributions with non-parametric models,

• **Why high-dimensional?** Because complex systems involve a large number of variables,

• **Why continuous?** Because in applications such as physics, engineering or finance variables are often continuous,

• **Why distributions?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),

• **Why non-parametric?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.
Learning high-dimensional probabilistic continuous models

• **Goal**: learning high dimensional continuous distributions with non-parametric models,

• **Why high-dimensional?** Because complex systems involve a large number of variables,

• **Why continuous?** Because in applications such as physics, engineering or finance variables are often continuous,

• **Why distributions?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),

• **Why non-parametric?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.

• **Challenge 1**: Various non-parametric models exist to estimate a density but they are limited to a few dimensions ($\sim 5$ variables),
Learning high-dimensional probabilistic continuous models

- **Goal**: learning high dimensional continuous distributions with non-parametric models,

- **Why high-dimensional?** Because complex systems involve a large number of variables,

- **Why continuous?** Because in applications such as physics, engineering or finance variables are often continuous,

- **Why distributions?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),

- **Why non-parametric?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.

- **Challenge 1**: Various non-parametric models exists to estimate a density but they are limited to a few dimensions (∼ 5 variables),

- **Solution**: Use of Probabilistic Graphical Models (PGM) to break the joint distribution into a product of conditional distributions of lesser dimensions.
Learning high-dimensional probabilistic continuous models

- **Goal**: learning high dimensional continuous distributions with non-parametric models,

- **Why high-dimensional?** Because complex systems involve a large number of variables,

- **Why continuous?** Because in applications such as physics, engineering or finance variables are often continuous,

- **Why distributions?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),

- **Why non-parametric?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.

- **Challenge 1**: Various non-parametric models exist to estimate a density but they are limited to a few dimensions (∼ 5 variables),

- **Solution**: Use of Probabilistic Graphical Models (PGM) to break the joint distribution into a product of conditional distributions of lesser dimensions.

- **Challenge 2**: We want a probabilistic model with a density from which we can sample points but continuous PGM are not satisfying,
Learning high-dimensional probabilistic continuous models

• **Goal**: learning high dimensional continuous distributions with non-parametric models,

• **Why high-dimensional?** Because complex systems involve a large number of variables,

• **Why continuous?** Because in applications such as physics, engineering or finance variables are often continuous,

• **Why distributions?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),

• **Why non-parametric?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.

• **Challenge 1**: Various non-parametric models exists to estimate a density but they are limited to a few dimensions (~ 5 variables),

• **Solution**: Use of Probabilistic Graphical Models (PGM) to break the joint distribution into a product of conditional distributions of lesser dimensions.

• **Challenge 2**: We want a probabilistic model with a density from which we can sample points but continuous PGM are not satisfying,

• **Solution**: Use of the Empirical Bernstein Copula to parameterize graphical models.
Copula Bayesian Networks (CBNs)
Bayesian Networks

- **Compact** representation of a joint probability distribution over a set of variables $\mathbf{X}$ using:

- A Directed Acyclic Graph (DAG),
- A set of Conditional Probability Distributions (CPD).

Discrete case: Conditional Probability Tables.
Continuous case: ???
• **Compact** representation of a joint probability distribution over a set of variables $\mathbf{X}$ using:
  - A Directed Acyclic Graph (DAG),

\[
\mathcal{I}_i(\mathcal{G}) = \{(X_i \perp \text{ND}_i|\text{Pa}_i)\}.
\]
Bayesian Networks

- **Compact** representation of a joint probability distribution over a set of variables $\mathbf{X}$ using:
  - A Directed Acyclic Graph (DAG),
  - A set of Conditional Probability Distributions (CPD).

$$f(\mathbf{X}) = \prod_{i=1}^{n} f(x_i | \text{pa}_i)$$
• **Compact** representation of a joint probability distribution over a set of variables $X$ using:
  
  - A Directed Acyclic Graph (DAG),
  - A set of Conditional Probability Distributions (CPD).

Discrete case: Conditional Probability Tables.
Bayesian Networks

- **Compact** representation of a joint probability distribution over a set of variables $X$ using:
  - A Directed Acyclic Graph (DAG),
  - A set of Conditional Probability Distributions (CPD).

\[
\begin{array}{c|c}
X_1 & f(X_1) \\
0 & 0.1 \\
1 & 0.9 \\
\end{array}
\]

\[
\begin{array}{c|c}
X_2 & f(X_2) \\
0 & 0.65 \\
1 & 0.35 \\
\end{array}
\]

\[
\begin{array}{c|ccc|c}
X_3 & X_1 & X_2 & f(X_3|X_1, X_2) \\
0 & 0 & 0 & 0.19 \\
0 & 0 & 1 & 0.85 \\
0 & 1 & 0 & 0.2 \\
0 & 1 & 1 & 0.8 \\
1 & 0 & 0 & 0.1 \\
1 & 0 & 1 & 0.9 \\
1 & 1 & 0 & 0.4 \\
1 & 1 & 1 & 0.6 \\
\end{array}
\]

\[
\begin{array}{c|ccc|c}
X_4 & X_3 & X_4 & f(X_4|X_3) \\
0 & 0 & 0 & 0.1 \\
0 & 0 & 1 & 0.9 \\
0 & 1 & 0 & 0.3 \\
1 & 1 & 0 & 0.7 \\
\end{array}
\]

**Discrete case**: Conditional Probability Tables.

**Continuous case**: ???
• **Discretization**:
  1. Limited to only a few bins for fast inference and learning algorithms.
  2. Which one do we choose to minimize the loss of information?
  3. How to a continuous model from there?
Bayesian Networks and continuous data

• **Discretization** :
  1. Limited to only a few bins for fast inference and learning algorithms.
  2. Which one do we chose to minimize the loss of information ?
  3. How to a continuous model from there ?

• **Linear Gaussian Bayesian Networks (LGBN)** Lauritzen et al.
  1989:  
  \[ f(y|x) = \mathcal{N}(y; \beta_0 + \sum_{i=1}^{k} \beta_i x_i, \sigma_y^2) \]
  1. **Good**: Fast inference and learning algorithms,
  2. **Bad**: Strong model assumptions (Gaussian),
Bayesian Networks and continuous data

- **Discretization**:
  1. Limited to only a few bins for fast inference and learning algorithms.
  2. Which one do we chose to minimize the loss of information?
  3. How to a continuous model from there?

- **Linear Gaussian Bayesian Networks (LGBN)** Lauritzen et al.
  
  1989: \[ f(y|x) = \mathcal{N}(y; \beta_0 + \sum_{i=1}^{k} \beta_i x_i, \sigma_y^2) \]
  
  1. **Good**: Fast inference and learning algorithms,
  2. **Bad**: Strong model assumptions (Gaussian),

- **Mixture models**: Langseth et al. 2012; Cortijo et al. 2016
  
  1. **Good**: Expressive models,
  2. **Bad**: Hard to learn
Copulas

\[ U = (U_1, \ldots, U_n) \]

Definition (Copula Nelsen 2007)

A copula function is a cumulative distribution function on \([0, 1]^n\):

\[ C(u_1, \ldots, u_n) = P(U_1 \leq u_1, \ldots, U_n \leq u_n) \]

with uniform one-dimensional marginals:

\[ C(1, \ldots, u_i, \ldots, 1) = u_i. \]

If \( C \) is absolutely continuous, a copula density function \( c \) exists:

\[ c(x) = \frac{\partial^n C}{\partial x_1 \cdots \partial x_n}(x_1, \ldots, x_n) \]
Copulas

- \( U = (U_1, \cdots, U_n) \), continuous random variable over \([0, 1]^n\),

Definition (Nelsen 2007)

A copula function is a cumulative distribution function on \([0, 1]^n\):

\[
C(u_1, \ldots, u_n) = P(U_1 \leq u_1, \ldots, U_n \leq u_n)
\]

with uniform one-dimensional marginals:

\[
C(1, \ldots, u_i, \ldots, 1) = u_i.
\]

- If \( C \) is absolutely continuous, a copula density function \( c \) exists:

\[
c(x) = \frac{\partial^n C}{\partial x_1 \cdots \partial x_n}(x_1, \ldots, x_n)
\]
• $U = (U_1, \ldots, U_n)$, **continuous** random variable over $[0, 1]^n$,

**Definition (Copula Nelsen 2007)**

A copula function is a cumulative distribution function on $[0, 1]^n$:

$$C(u_1, \ldots, u_n) = \mathbb{P}(U_1 \leq u_1, \ldots, U_n \leq u_n)$$

with **uniform** one-dimensional marginals:

$$C(1, \ldots, u_i, \ldots, 1) = u_i.$$
Copulas

- \( U = (U_1, \ldots, U_n) \), **continuous** random variable over \([0, 1]^n\),

**Definition (Copula Nelsen 2007)**

A copula function is a cumulative distribution function on \([0, 1]^n\) :

\[
C(u_1, \ldots, u_n) = P(U_1 \leq u_1, \ldots, U_n \leq u_n)
\]

with **uniform** one-dimensional marginals :

\[
C(1, \ldots, u_i, \ldots, 1) = u_i.
\]

- If \( C \) is absolutely **continuous**, a copula density function \( c \) exists :

\[
c(x) = \frac{\partial^n C}{\partial x_1 \cdots \partial x_n}(x_1, \ldots, x_n)
\]
**Sklar’s theorem**

<table>
<thead>
<tr>
<th>Theorem (Sklar, 1959)</th>
</tr>
</thead>
</table>

For any continuous distribution $F$ over $X_1, \ldots, X_n$, there exists a unique copula function $C$, such that:

$$F(x_1, \ldots, x_n) = C(F_1(x_1), \ldots, F_n(x_n))$$

Moreover, if $F$ is absolutely continuous,

$$f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n))$$

- Decomposition of the joint distribution into a copula function and a set of marginals: more freedom for modeling.
- $C$ encodes all the information about the dependencies between the variables: interesting for independence tests.
- $C$ becomes hard to model for high dimensions.
- Solution: use the BN framework over the copula function $\rightarrow$ Copula Bayesian Networks (CBNs) (Elidan 2010).
Theorem (Sklar, 1959)

For any continuous distribution \( F \) over \( X_1, \cdots, X_n \), there exists a unique copula function \( C \), such that:

\[
F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))
\]

Moreover, if \( F \) is absolutely continuous,

\[
f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n))
\]
## Sklar’s theorem

### Theorem (Sklar, 1959)

For any *continuous* distribution $F$ over $X_1, \cdots, X_n$, there exists a *unique* copula function $C$, such that:

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

- Decomposition of the joint distribution into a copula function and a set of marginals: more freedom for modeling.
- $C$ encodes all the information about the dependencies between the variables: interesting for independence tests.
- $C$ becomes hard to model for high dimensions.

Solution: use the BN framework over the copula function. → Copula Bayesian Networks (CBNs) (Elidan 2010)
Theorem (Sklar, 1959)

For any continuous distribution $F$ over $X_1, \cdots, X_n$, there exists a unique copula function $C$, such that:

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

Moreover, if $F$ is absolutely continuous,

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$
Theorem (Sklar, 1959)

For any continuous distribution \( F \) over \( X_1, \ldots, X_n \), there exists a unique copula function \( C \), such that:

\[
F(x_1, \ldots, x_n) = C(F_1(x_1), \ldots, F_n(x_n))
\]

Moreover, if \( F \) is absolutely continuous,

\[
f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)
\]

- Decomposition of the joint distribution into a copula function and a set of marginals: more freedom for modeling.
Sklar’s theorem

Theorem (Sklar, 1959)

For any continuous distribution $F$ over $X_1, \ldots, X_n$, there exists a unique copula function $C$, such that:

$$F(x_1, \ldots, x_n) = C(F_1(x_1), \ldots, F_n(x_n))$$

Moreover, if $F$ is absolutely continuous,

$$f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

- Decomposition of the joint distribution into a copula function and a set of marginals: more freedom for modeling.
- $C$ encodes all the information about the dependencies between the variables: interesting for independence tests.
Sklar’s theorem

**Theorem (Sklar, 1959)**

For any **continuous** distribution $F$ over $X_1, \cdots, X_n$, there exists a **unique** copula function $C$, such that:

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

Moreover, if $F$ is **absolutely** continuous,

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

- Decomposition of the joint distribution into a copula function and a set of marginals: **more freedom for modeling**.

- $C$ encodes all the information about the dependencies between the variables: **interesting for independence tests**.

⚠️ $C$ becomes hard to model for high dimensions.
**Theorem (Sklar, 1959)**

For any *continuous* distribution $F$ over $X_1, \cdots, X_n$, there exists a *unique* copula function $C$, such that:

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

Moreover, if $F$ is *absolutely* continuous,

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

- **Decomposition of the joint distribution into a copula function and a set of marginals:** more freedom for modeling.
- **$C$** encodes all the information about the dependencies between the variables: interesting for independence tests.

⚠️ **$C$** becomes hard to model for high dimensions.

- **Solution:** use the BN framework over the copula function → Copula Bayesian Networks (CBNs) (Elidan 2010)
Example: Gaussian copula

(a) Gaussian CDF

(b) Gaussian PDF

- \((X_1, X_2) \sim \mathcal{N}(\mu, \Sigma)\), with \(\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}\) and \(\Sigma = \begin{pmatrix} 1 & 0.45 \\ 0.45 & 0.25 \end{pmatrix}\)
Example: Gaussian copula

(a) Copula function

(b) Copula density function

• \((X_1, X_2) \sim \mathcal{N}(\mu, \Sigma)\), with \(\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}\) and \(\Sigma = \begin{pmatrix} 1 & 0.45 \\ 0.45 & 0.25 \end{pmatrix}\)

• \((U_1 = \Phi_{0,1}(X_1), U_2 = \Phi_{0, \frac{1}{2}}(X_2)) \sim C_{\mathcal{N}}(R)\) with \(R = \begin{pmatrix} 1 & 0.9 \\ 0.9 & 1 \end{pmatrix}\)
Definition (Copula Bayesian Network, Elidan 2010)

A Copula Bayesian Network (CBN) is a triplet $(G, \Theta_C, \Theta_f)$ which encodes a joint density $f(X)$ that factorizes over $G$:

$$f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

where $R_i(u_i|\pi_i) = c_i(u_i, \pi_i)c_i(\pi_i)$.

- Same graphical language than classical BNs (same independences)
- Classic algorithms can be adapted for structural learning.
Copula Bayesian Networks: definition

**Definition (Copula Bayesian Network, Elidan 2010)**

- $\mathcal{G}$: DAG over $X$,
- $\Theta_C$: set of (local) copula densities $c_i$,
- $\Theta_f$: set of marginal densities $f_i$.

A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$ which encodes a joint density $f(X)$ that factorizes over $\mathcal{G}$:

$$f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n)) \prod_{i=1}^n f_i(x_i)$$

where $R_i(u_i|\pi_i) = c_i(u_i, \pi_i) c_i(\pi_i)$.

- Same graphical language than classical BNs (same independences).
- Classic algorithms can be adapted for structural learning.
Copula Bayesian Networks: definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\mathbf{X}$,
- $\Theta_C$: set of (local) copula densities $c_i$,
- $\Theta_f$: set of marginal densities $f_i$.

A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$ which encodes a joint density $f(\mathbf{X})$ that factorizes over $\mathcal{G}$:

$$f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n))) = \prod_{i=1}^n f_i(x_i) \cdot R_i(u_i|\pi_i)$$

where $R_i(u_i|\pi_i) = c_i(u_i, \pi_i)$.

- Same graphical language than classical BNs (same independences)
- Classic algorithms can be adapted for structural learning.
Copula Bayesian Networks: definition

Definition (Copula Bayesian Network, Elidan 2010)

- $G$: DAG over $X$,
- $\Theta_C$: set of (local) copula densities $c_i$,
- $\Theta_f$: set of marginal densities $f_i$
Copula Bayesian Networks: definition

Definition (Copula Bayesian Network, Elidan 2010)

- \( G \) : DAG over \( X \),
- \( \Theta_C \) : set of (local) copula densities \( c_i \),
- \( \Theta_f \) set of marginal densities \( f_i \)

A Copula Bayesian Network (CBN) is a triplet \((G, \Theta_C, \Theta_f)\)
Copula Bayesian Networks: definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\mathbf{X}$,
- $\Theta_C$: set of (local) copula densities $c_i$,
- $\Theta_f$: set of marginal densities $f_i$

A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$ which encodes a joint density $f(\mathbf{X})$ that factorizes over $\mathcal{G}$:

$$f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i) \quad \text{(Sklar)}$$

$$= \prod_{i=1}^{n} R_i(F_i(x_i)|\mathcal{F}(\text{pa}_{X_i})) \cdot f_i(x_i)$$

where $R_i(u_i|\pi_i) = \frac{c_i(u_i, \pi_i)}{c_i(\pi_i)}$. 

• Same graphical language than classical BNs (same independences)
• Classic algorithms can be adapted for structural learning.
Copula Bayesian Networks: definition

**Definition (Copula Bayesian Network, Elidan 2010)**

- $\mathcal{G}$: DAG over $\mathbf{X}$,
- $\Theta_C$: set of (local) copula densities $c_i$,
- $\Theta_f$: set of marginal densities $f_i$

A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$ which encodes a joint density $f(\mathbf{X})$ that factorizes over $\mathcal{G}$:

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i) \quad \text{(Sklar)}$$

$$= \prod_{i=1}^{n} R_i(F_i(x_i)|\mathbf{F(pa}_X)) \cdot f_i(x_i)$$

where $R_i(u_i|\pi_i) = \frac{c_i(u_i, \pi_i)}{c_i(\pi_i)}$.

- Same graphical language than classical BNs (same independences)
Copula Bayesian Networks: definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\mathbf{X}$,
- $\Theta_C$ : set of (local) copula densities $c_i$,
- $\Theta_f$ set of marginal densities $f_i$

A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$ which encodes a joint density $f(\mathbf{X})$ that factorizes over $\mathcal{G}$:

$$f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i) \quad \text{(Sklar)}$$

$$= \prod_{i=1}^{n} R_i(F_i(x_i)|\mathbf{F}(\text{pa}_{X_i})) \cdot f_i(x_i)$$

where $R_i(u_i|\pi_i) = \frac{c_i(u_i, \pi_i)}{c_i(\pi_i)}$.

- Same graphical language than classical BNs (same independences)
- Classic algorithms can be adapted for structural learning.
Copula Bayesian Networks: example

\[ X_1, X_2, X_3, X_4 \]

\[ \Theta_c = \{ c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3) \} \]

\[ \Theta_f = \{ f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4) \} \]

\[ f(x_1, x_2, x_3, x_4) = \left[ R_1(F_1(x_1)) f_1(x_1) \right] \left[ R_2(F_2(x_2)) f_2(x_2) \right] \times \left[ R_3(F_3(x_3) | F_1(x_1), F_2(x_2)) f_3(x_3) \right] \times \left[ R_4(F_4(x_4) | F_3(x_3)) f_4(x_4) \right] \]

Parametric copulas: Gaussian, Student, Dirichlet, ...

Non-parametric copulas: Empirical Bernstein Copula (EBC)
Copula Bayesian Networks: example

\[(c_1(u_1) \equiv 1, f_1(x_1))\]
\[(c_2(u_2) \equiv 1, f_2(x_2))\]
\[(c_3(u_3, u_1, u_2), f_3(x_3))\]
\[(c_4(u_4, u_3), f_4(x_4))\]
Copula Bayesian Networks: example

\[
(c_1(u_1) \equiv 1, f_1(x_1)) \quad X_1 \quad (c_2(u_2) \equiv 1, f_2(x_2)) \quad X_2
\]

\[
(c_3(u_3, u_1, u_2), f_3(x_3)) \quad X_3
\]

\[
(c_4(u_4, u_3), f_4(x_4)) \quad X_4
\]

- \( \Theta_C = \{ c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3) \} \)
Copula Bayesian Networks: example

\[ (c_1(u_1) \equiv 1, f_1(x_1)) \]

\[ (c_2(u_2) \equiv 1, f_2(x_2)) \]

\[ (c_3(u_3, u_1, u_2), f_3(x_3)) \]

\[ (c_4(u_4, u_3), f_4(x_4)) \]

- \( \Theta_C = \{ c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3) \} \)
- \( \Theta_f = \{ f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4) \} \)
Copula Bayesian Networks: example

\[(c_1(u_1) \equiv 1, f_1(x_1))\]  
\[X_1\]  
\[\rightarrow\]

\[(c_2(u_2) \equiv 1, f_2(x_2))\]  
\[X_2\]  
\[\rightarrow\]

\[(c_3(u_3, u_1, u_2), f_3(x_3))\]  
\[X_3\]  
\[\rightarrow\]

\[(c_4(u_4, u_3), f_4(x_4))\]  
\[X_4\]

- \(\Theta_C = \{c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3)\}\)
- \(\Theta_f = \{f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4)\}\)
- \(f(x_1, x_2, x_3, x_4) = [R_1(F_1(x_1))f_1(x_1)][R_2(F_2(x_2))f_2(x_2)]\)  
  \[\times [R_3(F_3(x_3)|F_1(x_1), F_2(x_2))f_3(x_3)]\)  
  \[\times [R_4(F_4(x_4)|F_3(x_3))f_4(x_4)]\)
Copula Bayesian Networks: example

\[
(c_1(u_1) \equiv 1, f_1(x_1)) \quad X_1 \quad (c_2(u_2) \equiv 1, f_2(x_2)) \quad X_2
\]

\[
(c_3(u_3, u_1, u_2), f_3(x_3)) \quad X_3
\]

\[
(c_4(u_4, u_3), f_4(x_4)) \quad X_4
\]

- \( \Theta_C = \{ c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3) \} \)
- \( \Theta_f = \{ f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4) \} \)
- \( f(x_1, x_2, x_3, x_4) = [R_1(F_1(x_1))f_1(x_1)][R_2(F_2(x_2))f_2(x_2)] \)
  \[ \times [R_3(F_3(x_3)|F_1(x_1), F_2(x_2))f_3(x_3)] \]
  \[ \times [R_4(F_4(x_4)|F_3(x_3))f_4(x_4)] \]
- Parametric copulas: Gaussian, Student, Dirichlet, …
Copula Bayesian Networks: example

\[(c_1(u_1) \equiv 1, f_1(x_1)) \quad X_1 \quad X_3 \quad (c_2(u_2) \equiv 1, f_2(x_2)) \quad X_2\]

\[(c_3(u_3, u_1, u_2), f_3(x_3)) \quad X_3 \quad X_4 \quad (c_4(u_4, u_3), f_4(x_4))\]

- \(\Theta_C = \{c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3)\}\)
- \(\Theta_f = \{f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4)\}\)
- \(f(x_1, x_2, x_3, x_4) = [R_1(F_1(x_1))f_1(x_1)][R_2(F_2(x_2))f_2(x_2)]\)
  \(\times [R_3(F_3(x_3)|F_1(x_1), F_2(x_2))f_3(x_3)]\)
  \(\times [R_4(F_4(x_4)|F_3(x_3))f_4(x_4)]\)

- Parametric copulas: Gaussian, Student, Dirichlet, ...
- Non-parametric copulas: Empirical Bernstein Copula (EBC)
Non-parametric estimation: empirical Bernstein copula

- Sample $\mathcal{D} = \{x[1], \ldots, x[m]\}$ → Copula sample $\mathcal{C} = \{u[1], \ldots, u[m]\}$
  with $u[m] = (u_1[m], \ldots, u_n[m])$, $u_i[m] = F_i(x_i[m])$
Non-parametric estimation: empirical Bernstein copula

- Sample $D = \{x[1], \ldots, x[m]\} \rightarrow$ Copula sample $C = \{u[1], \ldots, u[m]\}$ with $u[m] = (u_1[m], \ldots, u_n[m])$, $u_i[m] = F_i(x_i[m])$

- Empirical copula:
  $$\hat{C}_m(u) = \frac{1}{m} \sum_{j=1}^{m} \prod_{i=1}^{n} 1\{U_i[j] \leq u_i\}.$$
Non-parametric estimation: empirical Bernstein copula

- Sample $\mathcal{D} = \{x[1], \ldots, x[m]\} \rightarrow$ Copula sample $\mathcal{C} = \{u[1], \ldots, u[m]\}$ with $u[m] = (u_1[m], \ldots, u_n[m])$, $u_i[m] = F_i(x_i[m])$

- Empirical copula:
  \[
  \hat{C}_m(u) = \frac{1}{m} \sum_{j=1}^{m} \prod_{i=1}^{n} 1\{U_i[j] \leq u_i\}.
  \]

- Bernstein polynomial:
  \[
  B_{v,m}(u) = \binom{m}{v} u^v (1 - u)^{m-v}
  \]
Non-parametric estimation: empirical Bernstein copula

- Sample $\mathcal{D} = \{x[1], \ldots, x[m]\} \rightarrow$ Copula sample $\mathcal{C} = \{u[1], \ldots, u[m]\}$ with $u[m] = (u_1[m], \ldots, u_n[m]), u_i[m] = F_i(x_i[m])$

- Empirical copula:
  \[
  \hat{C}_m(u) = \frac{1}{m} \sum_{j=1}^{m} \prod_{i=1}^{n} \mathbb{1}\{U_i[j] \leq u_i\}.
  \]

- Bernstein polynomial:
  \[
  B_{v,m}(u) = \binom{m}{v} u^v (1 - u)^{m-v}
  \]

- Empirical Bernstein copula (EBC) $\hat{C}_B$:
  \[
  \hat{C}^B_{K,m}(u) = \sum_{v_1=0}^{K} \cdots \sum_{v_n=0}^{K} \hat{C}_m\left(\frac{v_1}{K}, \ldots, \frac{v_n}{K}\right) \prod_{i=1}^{n} B_{v_i,K}(u_i),
  \]
• Sample $\mathcal{D} = \{x[1], \ldots, x[m]\} \rightarrow$ Copula sample $\mathcal{C} = \{u[1], \ldots, u[m]\}$ with $u[m] = (u_1[m], \ldots, u_n[m])$, $u_i[m] = F_i(x_i[m])$

• Empirical copula:

\[
\hat{C}_m(u) = \frac{1}{m} \sum_{j=1}^{m} \prod_{i=1}^{n} 1\{U_i[j] \leq u_i\}.
\]

• Bernstein polynomial :

\[
B_{v,m}(u) = \binom{m}{v} u^v (1-u)^{m-v}
\]

• Empirical Bernstein copula (EBC) $\hat{C}_B$ :

\[
\hat{C}^B_{K,m}(u) = \frac{1}{m} \sum_{i=1}^{m} \prod_{j=1}^{n} l_{r_j[i], s_j[i]}(u_j)
\]

with $r_j[i] = \lceil Ku_j[i] \rceil$, $s_j[i] = K - r_j[i] + 1$ and $l_{\alpha,\beta}$ the cumulative function of beta distribution,
Non-parametric estimation: empirical Bernstein copula

- Sample $\mathcal{D} = \{x[1], \ldots, x[m]\} \rightarrow$ Copula sample $\mathcal{C} = \{u[1], \ldots, u[m]\}$ with $u[m] = (u_1[m], \ldots, u_n[m])$, $u_i[m] = F_i(x_i[m])$

- Empirical copula:
  \[
  \hat{C}_m(u) = \frac{1}{m} \sum_{j=1}^{m} \prod_{i=1}^{n} 1\{U_i[j] \leq u_i\}.
  \]

- Bernstein polynomial:
  \[
  B_{\nu, m}(u) = \binom{m}{\nu} u^\nu (1 - u)^{m-\nu}
  \]

- Empirical Bernstein copula (EBC) $\hat{C}_B$:
  \[
  \hat{C}_{K,m}(u) = \frac{1}{m} \sum_{i=1}^{m} \prod_{j=1}^{n} I_{r_j[i],s_j[i]}(u_j)
  \]
  with $r_j[i] = \lceil Ku_j[i]\rceil$, $s_j[i] = K - r_j[i] + 1$ and $I_{\alpha,\beta}$ the cumulative function of beta distribution,

- Empirical Bernstein copula (EBC) density $\hat{c}_B$ by differentiation:
  \[
  \hat{c}_B(u) = \frac{1}{m} \sum_{i=1}^{m} \prod_{j=1}^{n} \beta_{r_j[i],s_j[i]}(u_j)
  \]
Non-parametric estimation: empirical Bernstein copula

(a) Gaussian sample
Non-parametric estimation: empirical Bernstein copula

(a) Gaussian copula density sample
Non-parametric estimation: empirical Bernstein copula

(a) Gaussian copula density

(b) Gaussian copula density sample
Non-parametric estimation: empirical Bernstein copula

(a) Gaussian copula density

(b) Bernstein copula: $m = 10^2$
Non-parametric estimation: empirical Bernstein copula

(a) Gaussian copula density

(b) Bernstein copula: $m = 10^3$
Non-parametric estimation: empirical Bernstein copula

(a) Gaussian copula density

(b) Bernstein copula: $m = 10^4$
The otagrum module
Two similar libraries (C++, python wrappers, open source):

- OpenTURNS deals with copulas and continuous distributions (available on GitHub, pip and conda).
- aGrUM deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: otagrum.

What does it contain?

- A CBN class,
- Several learning algorithms,
- A detailed documentation.

Where to find it?

- Module: openturns/otagrum (GitHub)
- Experiments: MLasserre/otagrum-experiments (GitHub)
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNs** deals with copulas and continuous distributions (available on GitHub, pip and conda).

A module to rule them all: **otagrum**.

What does it contain?

- A CBN class,
- Several learning algorithms,
- A detailed documentation.

Where to find it?

- Module: openturns/otagrum (GitHub)
- Experiments: MLasserre/otagrum-experiments (GitHub)
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).
Two similar libraries (C++, python wrappers, open source):

- **OpenTurns** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

*A module to rule them all: otagrum.*
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

*A module to rule them all: otagrum.*

**What does it contain?**
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

*A module to rule them all*: **otagrum**.

What does it contain?
- A CBN class,
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

*A module to rule them all*: **otagrum**.

**What does it contain?**

- A CBN class,
- Several learning algorithms,
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

**What does it contain?**
- A CBN class,
- Several learning algorithms,
- A detailed documentation.
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

**What does it contain ?**

- A CBN class,
- Several learning algorithms,
- A detailed documentation.

**Where to find it ?**
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

*A module to rule them all*: **otagrum**.

**What does it contain?**

- A CBN class,
- Several learning algorithms,
- A detailed documentation.

**Where to find it?**

- Module: openturns/otagrum (GitHub)
Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

**What does it contain?**

- A CBN class,
- Several learning algorithms,
- A detailed documentation.

**Where to find it?**

- Module: openturns/otagrum (GitHub)
- Experiments: MLasserre/otagrum-experiments (GitHub)
**otagrum: installation**

- **Online website**: [https://openturns.github.io/otagrum/master/index.html](https://openturns.github.io/otagrum/master/index.html)

- Can be easily installed using conda:
  
  `$ conda install -c conda-forge otagrum`

- Or manually to have the development version.
Using OTaGrUM: The wine data set

**Importing modules**

```python
import openturns as ot
import openturns.viewer as otv

import pyAgrum as gum
import pyAgrum.lib.notebook as gnb
import otagrum as otagr
```

**Loading data**

```python
data_ref = ot.Sample.ImportFromTextFile('winequality-red.csv', ';')
```
Structure learning with CBIC algorithm

Entree [3]:

```python
learner = otagr.TabuList(data_ref, 2, 10, 2)  # Creating a TabuList learner
cbic_dag = learner.learnDAG()                # Learning DAG
gnb.showDot(cbic_dag.toDot())
```
structure learning with CPC algorithm

\begin{verbatim}
Entrée [4]:
learner = otagr.ContinuousPC(data_ref, 4, 0.05)  # Using a CPC learner
cpc_dag = learner.learnDAG()  # Learning DAG
gnml.showDot(cpc_dag.toDot())
\end{verbatim}
**Structure learning with CMIIC algorithm**

```python
Entrée [5]:
learner = otagr.ContinuousMIIC(data_ref) # Using a CMIIC learner
learner.setAlpha(0.04) # Setting the value of alpha
cmiic_dag = learner.learnDAG() # Learning DAG
gnb.showDot(cmiic_dag.toDot())
```
otagrum: an example of use

Parameter learning

```python
Entré [7]:

cpc_cbn = otagr.ContinuousBayesianNetworkFactory(ot.KernelSmoothing(ot.Histogram()),
    ot.BernsteinCopulaFactory(),
    cpc_dag,
    0.05,
    4,
    False).build(data_ref)
```
Sampling the CBN

Entrée [9]:
```python
sample = cpc_cbn.getSample(1000)
ot.VisualTest.DrawPairs(sample.getMarginal([0,1]))
```

Out[9]:
![Scatter plot with fixed acidity vs. volatile acidity](attachment:image.png)
Structure learning for CBNs
Learning algorithms

- CPC, a continuous PC algorithm based on an independence test using Hellinger distance:

- CMIIC, an algorithm based on information theory:

- Improvement of the state of the art algorithm (CBIC) by using mutual information to speed up the calculations.
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

→ 1. A reference structure is chosen : ALARM or random,
   2. Copulas are parametrized : Gaussian, Student or Dirichlet,
   3. Samples are generated from the CBN : forward-sampling,
   4. A structure is learned from the generated data,
   5. Structural scores are computed : F-score et SHD.

Number of nodes : $n$  $\implies$  Number of arcs : $\lfloor 1.2 \times n \rfloor$
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

1. A reference structure is chosen: ALARM or random,
→ 2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

1. A reference structure is chosen: ALARM or random,
   → 2. Copulas are parametrized: Gaussian, **Student** or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
→ 5. Structural scores are computed: F-score et SHD.

- F-score: skeleton (undirected structure)
  - Skeleton perfectly retrieved: $F$-score = 1
- Structural Hamming Distance (SHD): CPDAG (skeleton + v-structures)
  - CPDAG perfectly retrieved: $\text{SHD} = 0$
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.

- F-score: skeleton (undirected structure)
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: **F-score** et SHD.

- **F-score**: skeleton (undirected structure)
  - Skeleton perfectly retrieved: $F\text{-}\text{score} = 1$
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed: F-score et SHD.

- F-score: skeleton (undirected structure)
  - Skeleton perfectly retrieved: F-score = 1

- Structural Hamming Distance (SHD): CPDAG (skeleton + v-structures)
Comparison method

1. A reference structure is chosen: ALARM or random,
2. Copulas are parametrized: Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN: forward-sampling,
4. A structure is learned from the generated data,
→ 5. Structural scores are computed: F-score et \(\text{SHD} \).

- **F-score**: skeleton (undirected structure)
  - Skeleton perfectly retrieved: \(F\text{-score} = 1\)

- **Structural Hamming Distance (SHD)**: CPDAG (skeleton + v-structures)
  - CPDAG perfectly retrieved: \(\text{SHD} = 0\)
F-score evolution for CBIC, CPC, G-CMIIC and B-CMIIC methods with respect to the sample size. For a given size, the results are averaged over 5 different samples generated from the ALARM structure.
**SHD evolution : ALARM structure**

**SHD evolution for CBIC, CPC, G-MIIC and B-MIIC methods with respect to the sample size.** For a given size, the results are averaged over 5 different samples generated from the ALARM structure.
F-score evolution for CBIC, CPC, G-MIIC and B-MIIC methods with respect to the dimension of the random structures. The results are averaged over 2 random structures of same dimension and over 5 different samples of size $m = 10^4$. 

(a) Gaussian case  
(b) Student case  
(c) Dirichlet case
SHD evolution for CBIC, CPC, G-CMIIC and B-CMIIC methods with respect to the dimension of the random structure. The results are averaged over 2 different structures of same dimension and over 5 different samples of size $m = 10^4$. 
Learning time in seconds for CBIC, CPC, G-CMIIC and B-CMIIC with respect to the dimension of the random structures. The results are averaged over 2 different random structures of same dimension and over 5 different samples of size $m = 10^4$. 

**Temporal complexity**

(a) Gaussian case  
(b) Student case  
(c) Dirichlet case
Thank you for your attention!


