Probabilistic affine forms
-
A first step towards the static analysis of probabilistic numerical programs.

Olivier Bouissou - Eric Goubault - Sylvie Putot

Séminaire LMeASI - 03 novembre 2010

Origin of this work.

- CPP project funded by the ANR, started in October 2009.

- *Goal of the project*: study the use of probabilistic information within the static analysis of numerical programs.
    - Probabilistic abstract domains.
    - Probabilistic programs.

- We have mainly considered probabilistic versions of intervals and affine sets.

Introductive example.

- Hipass order 2 linear filter.
- Input values are usually generated by sensors (here simulated by the `inputsignal` function).
- Number of iterations (N) is usually not known.
- *Output*: yn.

```
float inputsignal(int i) {
  return sin(F*i);
}

int main() {
  float xnm2,xnm1,xn,ynm2,ynm1,yn;
  int i;

  xnm2 = inputsignal(-2);
  xnm1 = inputsignal(-1);
  xn = inputsignal(0);
  ynm2 = 0;
  ynm1 = 0;

  for (i=1;i<=N;i++) {
    yn = (2*(c*c-1)*ynm1-
        (c*c-sqrt(2)*c+1)*ynm2+
        c*c*xn-2*c*c*xnm1+
        c*c*xnm2)/(c*c+sqrt(2)*c+1);
    ynm2 = ynm1;
    ynm1 = yn;
    xnm2 = xnm1;
    xnm1 = xn;
    xn = inputsignal(i);
  }
  return 0;
}
```

Introductive example.

- Hipass order 2 linear filter.
- Input values are usually generated by sensors (here simulated by the `inputsignal` function).
- *Inputs are usually not known.*
- Number of iterations (N) is usually not known.
- *Output*: yn.
- *Need to bound the values of* yn.

```
float inputsignal(int i) {
  return sin(F*i); [-1,1];
}

int main() {
  float xnm2,xnm1,xn,ynm2,ynm1,yn;
  int i;

  xnm2 = inputsignal(-2);
  xnm1 = inputsignal(-1);
  xn = inputsignal(0);
  ynm2 = 0;
  ynm1 = 0;

  for (i=1;i<=N;i++) {
    yn = (2*(c*c-1)*ynm1-
        (c*c-sqrt(2)*c+1)*ynm2+
        c*c*xn-2*c*c*xnm1+
        c*c*xnm2)/(c*c+sqrt(2)*c+1);
    ynm2 = ynm1;
    ynm1 = yn;
    xnm2 = xnm1;
    xnm1 = xn;
    xn = inputsignal(i);
  }
  return 0;
}
```

Static analysis in 10 seconds.

- ***Goal of static analysis***: compute the smallest possible $D \subseteq \mathbb{R}^n$ such that for all execution of the program, the values of the variables remain within D.
- Classical scheme: program $\longrightarrow$ control flow graph $\longrightarrow$ semantic equations

```
float inputsignal(int i) {
  return [-1,1];
}

int main() {
  float xnm2,xnm1,xn,ynm2,ynm1,yn;
  int i;

  xnm2 = inputsignal(-2);
  xnm1 = inputsignal(-1);
  xn = inputsignal(0);
  ynm2 = 0;
  ynm1 = 0;

  for (i=1;i<=N;i++) {
    yn = (2*(c*c-1)*ynm1-
          (c*c-sqrt(2)*c+1)*ynm2+
          c*c*xn-2*c*c*xnm1+
          c*c*xnm2)/(c*c+sqrt(2)*c+1);
    ynm2 = ynm1;
    ynm1 = yn;
    xnm2 = xnm1;
    xnm1 = xn;
    xn = inputsignal(i);
  }
  return 0;
}
```

Static analysis in 10 seconds.

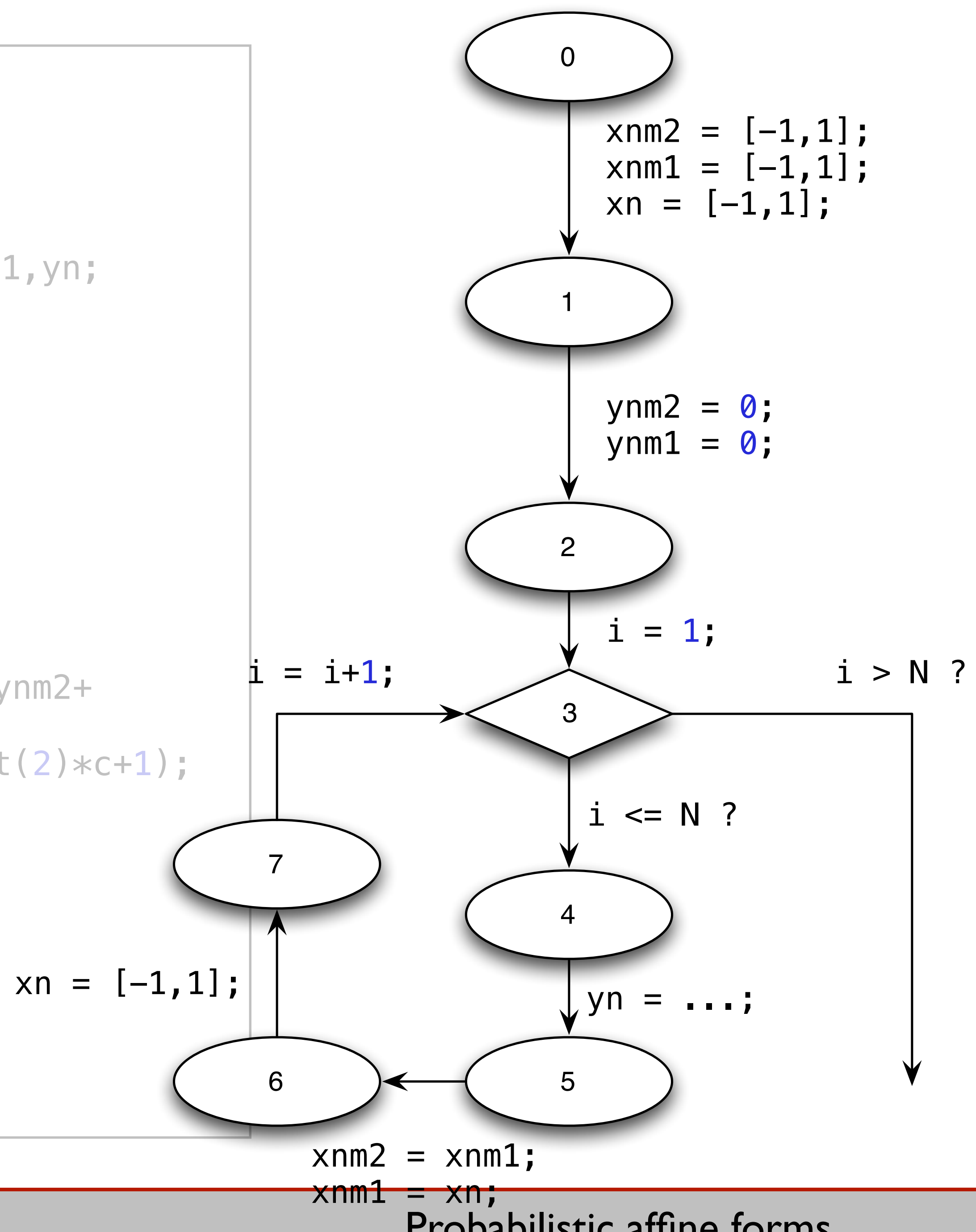
- *Goal of static analysis*: compute the smallest possible $D \subseteq \mathbb{R}^n$ such that for all execution of the program, the values of the variables remain within D.
- Classical scheme: program $\longrightarrow$ control flow graph $\longrightarrow$ semantic equations

```
float inputsignal(int i) {
  return [-1,1];
}

int main() {
  float xnm2,xnm1,xn,ynm2,ynm1,yn;
  int i;

  xnm2 = inputsignal(-2);
  xnm1 = inputsignal(-1);
  xn = inputsignal(0);
  ynm2 = 0;
  ynm1 = 0;

  for (i=1;i<=N;i++) {
    yn = (2*(c*c-1)*ynm1-
          (c*c-sqrt(2)*c+1)*ynm2+
          c*c*xn-2*c*c*xnm1+
          c*c*xnm2)/(c*c+sqrt(2)*c+1);
    ynm2 = ynm1;
    ynm1 = yn;
    xnm2 = xnm1;
    xnm1 = xn;
    xn = inputsignal(i);
  }
  return 0;
}
```



Static analysis in 10 seconds.

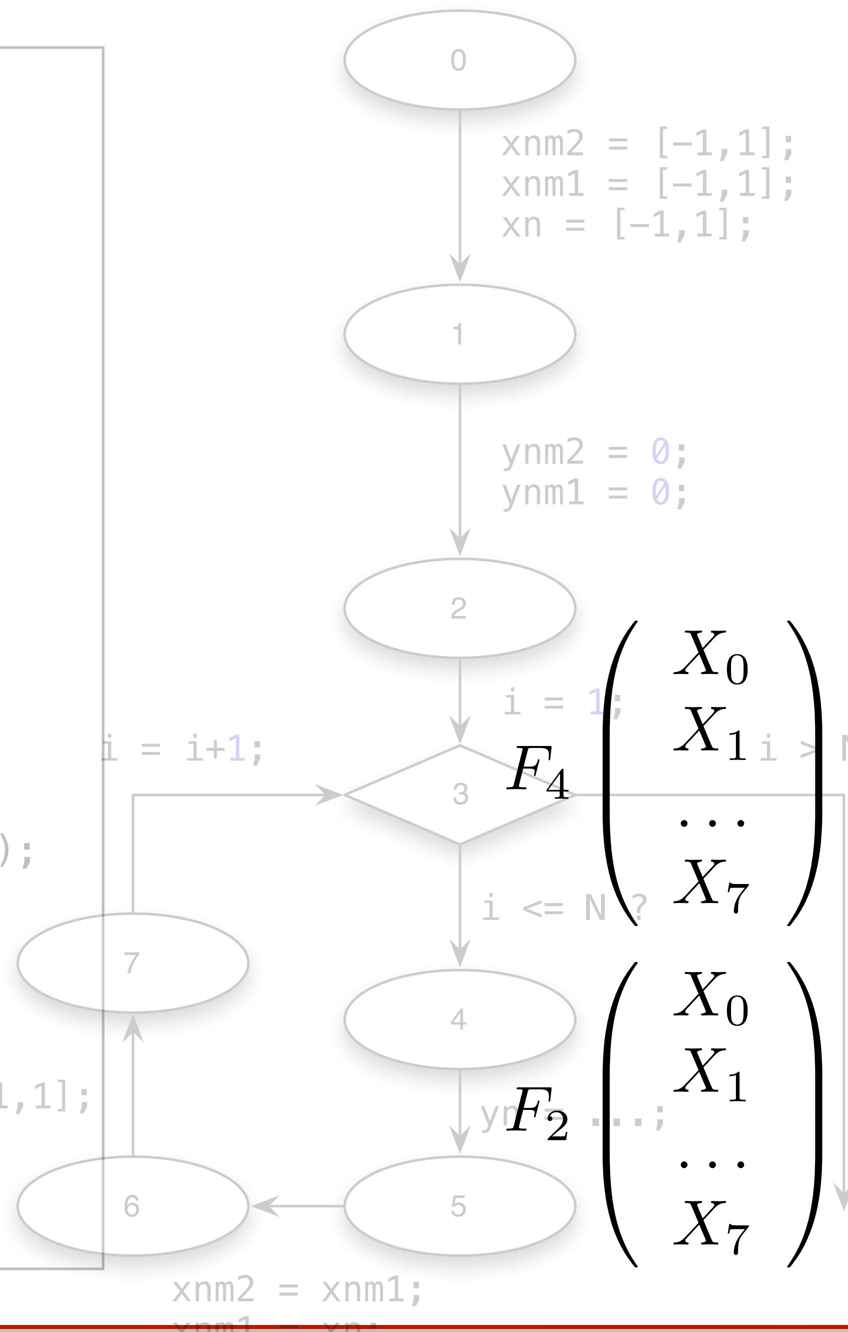
- *Goal of static analysis*: compute the smallest possible $D \subseteq \mathbb{R}^n$ such that for all execution of the program, the values of the variables remain within D.
- Classical scheme: program $\longrightarrow$ control flow graph $\longrightarrow$ semantic equations

```
float inputsignal(int i) {
  return [-1,1];
}

int main() {
  float xnm2,xnm1,xn,ynm2,ynm1,yn;
  int i;

  xnm2 = inputsignal(-2);
  xnm1 = inputsignal(-1);
  xn = inputsignal(0);
  ynm2 = 0;
  ynm1 = 0;

  for (i=1;i<=N;i++) {
    yn = (2*(c*c-1)*ynm1-
          (c*c-sqrt(2)*c+1)*ynm2+
          c*c*xn-2*c*c*xnm1+
          c*c*xnm2)/(c*c+sqrt(2)*c+1);
    ynm2 = ynm1;
    ynm1 = yn;
    xnm2 = xnm1;
    xnm1 = xn;
    xn = inputsignal(i);
  }
  return 0;
}
```



$$\begin{pmatrix} X_0 \\ X_1 \\ \dots \\ X_7 \end{pmatrix} = F \begin{pmatrix} X_0 \\ X_1 \\ \dots \\ X_7 \end{pmatrix}$$

$$F_4 \begin{pmatrix} X_0 \\ X_1 \\ \dots \\ X_7 \end{pmatrix} = X_4\left[\mathtt{yn} \mapsto 2 * (c * c - 1) * \mathtt{ynm1} \dots\right]$$

$$F_2 \begin{pmatrix} X_0 \\ X_1 \\ \dots \\ X_7 \end{pmatrix} = X_2\left[\mathtt{i} \mapsto 1\right] \cup X_7\left[\mathtt{i} \mapsto \mathtt{i} + 1\right]$$

Static analysis in 10 (more) seconds.

- All we need to do now is solve the semantic equations.
- Classical scheme: abstract domain ⟶ Kleene iteration ⟶ widening.

- The abstract domain must have:
  - order theoretic operations (union, intersection) to handle the control flow.

$$X_2[\mathtt{i} \mapsto 1] \cup X_7[\mathtt{i} \mapsto \mathtt{i}+1]$$

  - evaluations of numerical operations to handle numerical expressions in the program.

$$X_4[\mathtt{yn} \mapsto 2*(c*c-1)*\mathtt{ynm1}-(c*c-\sqrt{2}*c+1)*\mathtt{ynm2}+c*c*\mathtt{xn}-\ldots]$$

- With soundness conditions w.r.t. the conrete semantics.

Solving the fixpoint equation: with intervals.

- We chose the interval abstract domain.

$$X_i = \left\{ x \mapsto [\underline{x}, \overline{x}],\ yn \mapsto [\underline{yn}, \overline{yn}],\ \ldots \right\}$$

- We unroll the loop 30 times and replace each arithmetic operation by its sound interval version.

```
float inputsignal(int i) {
  return [-1,1];
}

int main() {
  float xnm2,xnm1,xn,ynm2,ynm1,yn;
  int i;

  xnm2 = inputsignal(-2);
  xnm1 = inputsignal(-1);
  xn = inputsignal(0);
  ynm2 = 0;
  ynm1 = 0;

  for (i=1;i<=N;i++) {
    yn = (2*(c*c-1)*ynm1-
          (c*c-sqrt(2)*c+1)*ynm2+
          c*c*xn-2*c*c*xnm1+
          c*c*xnm2)/(c*c+sqrt(2)*c+1);
    ynm2 = ynm1;
    ynm1 = yn;
    xnm2 = xnm1;
    xnm1 = xn;
    xn = inputsignal(i);
  }
  return 0;
}
```

$$yn \in [-9.79528957, 9.79528957].10^9$$

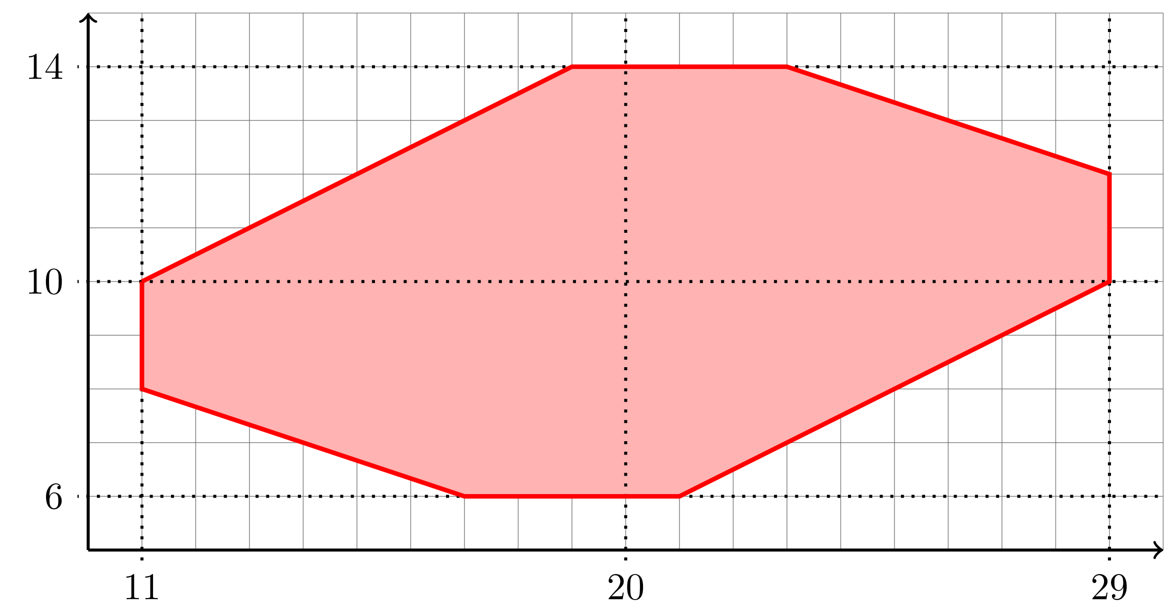
The affine set abstract domain

$$X_i = \left\{ \begin{array}{ccc} x & \mapsto & \alpha_0^x + \sum_{i=1}^m \alpha_i^x \epsilon_i \\ y & \mapsto & \alpha_0^y + \sum_{i=1}^m \alpha_i^y \epsilon_i \\ & \dots & \end{array} \middle| \alpha_i^v \in \mathbb{R},\ \epsilon_i \in [-1,1] \right\}$$

$$\gamma(X_i) = \left\{ (x, y, \dots) \in \mathbb{R}^m \middle| \exists \epsilon_1, \dots, \epsilon_n \in [-1,1]^n,\ \begin{array}{c} x = \alpha_0^x + \sum_{i=1}^m \alpha_i^x \epsilon_i \\ y = \alpha_0^y + \sum_{i=1}^m \alpha_i^y \epsilon_i \\ \dots \end{array} \right\}$$

$$x = 20 - 4\epsilon_1 + 2\epsilon_3 + 3\epsilon_4$$

$$y = 10 - 2\epsilon_1 + \epsilon_2 - \epsilon_4$$



Solving the fixpoint equation: with affine sets.

Confidence Proof Probabilities

The affine set arithmetic: linear operations

$$ax + by + c = (\alpha_0^x + \alpha_0^y + c) + \sum_{i=1}^{m} (\alpha_i^x + \alpha_i^y)\epsilon_i$$

Linear operations are *exact*.

Affectation creates a new noise symbol.

`x=[a,b];` $\longrightarrow$ $x = \dfrac{a+b}{2} + \dfrac{b-a}{2}\epsilon_{m+1}$

$$x = 20 - 4\epsilon_1 + 2\epsilon_3 + 3\epsilon_4$$
$$y = 10 - 2\epsilon_1 + \epsilon_2 - \epsilon_4$$

```
t=[1,5];
z=x+3y+t;
```

$\longrightarrow$

$$x = 20 - 4\epsilon_1 + 2\epsilon_3 + 3\epsilon_4$$
$$y = 10 - 2\epsilon_1 + \epsilon_2 - \epsilon_4$$
$$z = 53 - 10\epsilon_1 + 3\epsilon_2 + 2\epsilon_3 + 2\epsilon_5$$
$$t = 3 + 2\epsilon_5$$

Confidence Proof Probabilities

The affine set arithmetic: non-linear operations

$$x * y = (\alpha_0^x * \alpha_0^y) + \sum_{i=1}^{m} (\alpha_0^x \alpha_i^y + \alpha_0^y \alpha_i^x)\epsilon_i + \sum_{1 \le i,j \le m} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j$$

Non-linear terms are approximated in a new noise symbol.

$$\sum_{1 \le i,j \le m} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j \subseteq \left(\sum_{i=1}^{m} |\alpha_i^x| \cdot \sum_{i=1}^{m} \left|\alpha_i^j\right| \right) \eta_1$$

$$x = 20 - 4\epsilon_1 + 2\epsilon_3 + 3\epsilon_4$$

$$y = 10 - 2\epsilon_1 + \epsilon_2 - \epsilon_4$$

```
t=[1,5];
z=x*t;
```

$\longrightarrow$

$$x = 20 - 4\epsilon_1 + 2\epsilon_3 + 3\epsilon_4$$

$$y = 10 - 2\epsilon_1 + \epsilon_2 - \epsilon_4$$

$$z = 60 - 12\epsilon_1 + 6\epsilon_3 + 9\epsilon_4 + 40\epsilon_5 + 18\eta_1$$

$$t = 3 + 2\epsilon_5$$

Solving the fixpoint equation: with affine sets.

- We chose the affine set abstract domain.
- We unroll the loop 30 times and replace each arithmetic operation by its sound affine set version.

```
float inputsignal(int i) {
  return [-1,1];
}

int main() {
  float xnm2,xnm1,xn,ynm2,ynm1,yn;
  int i;

  xnm2 = inputsignal(-2);
  xnm1 = inputsignal(-1);
  xn = inputsignal(0);
  ynm2 = 0;
  ynm1 = 0;

  for (i=1;i<=N;i++) {
    yn = (2*(c*c-1)*ynm1-
          (c*c-sqrt(2)*c+1)*ynm2+
          c*c*xn-2*c*c*xnm1+
          c*c*xnm2)/(c*c+sqrt(2)*c+1);
    ynm2 = ynm1;
    ynm1 = yn;
    xnm2 = xnm1;
    xnm1 = xn;
    xn = inputsignal(i);
  }
  return 0;
}
```

$$yn \in [-2.33900235, 2.33900235]$$

Some remarks about the inputs.

Confidence Proof Probabilities

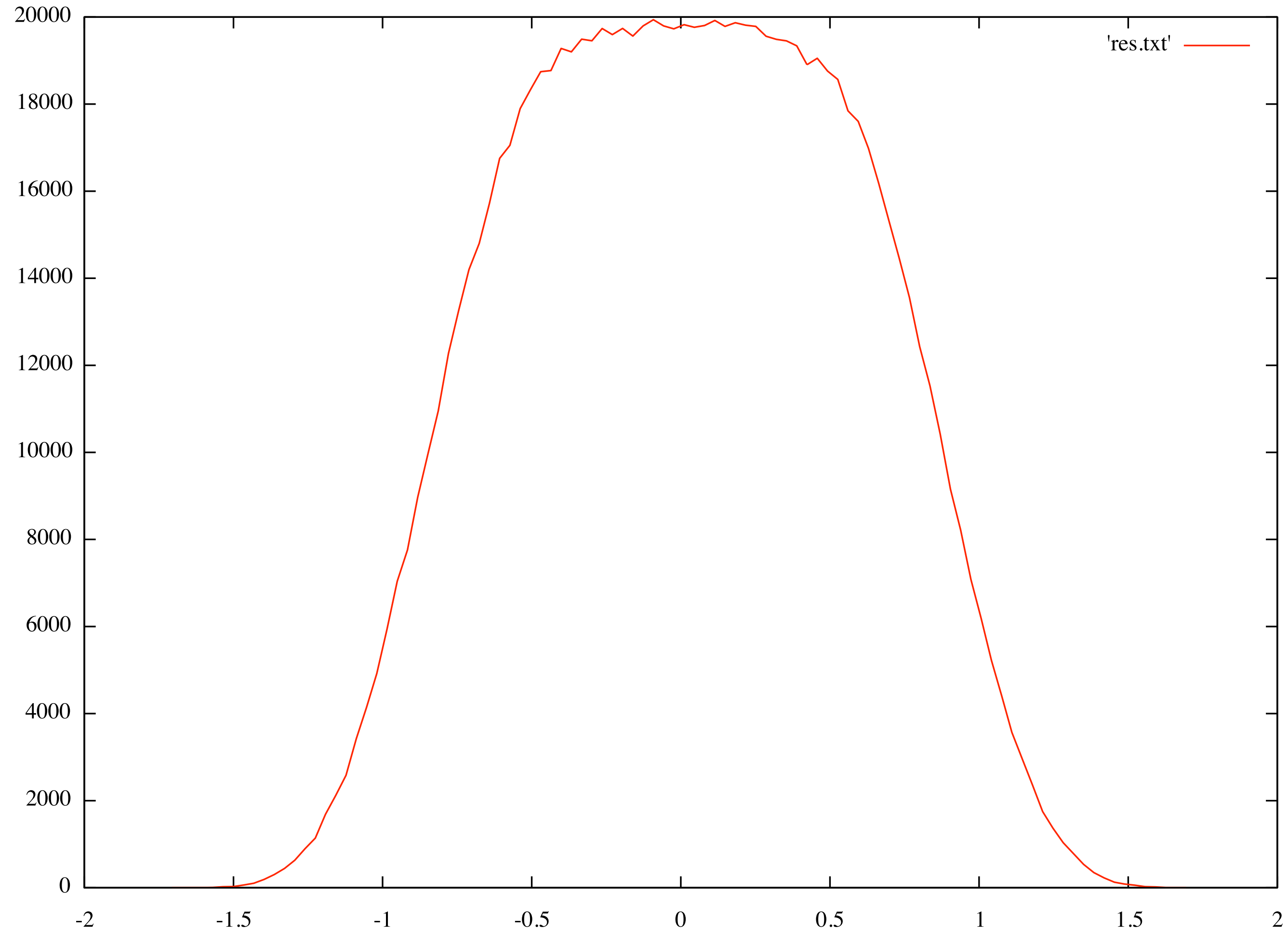
- What does it mean: `x=[-1,1]` ?
- At each loop iterate, a new value of x can be non-deterministically chosen between -1 and 1.
- It is similar to a two-persons game with malicious opponent.
- What if we have a probabilistic opponent ?

Some remarks about the inputs.

- What does it mean: $x=$ `[-1,1]` ?
- At each loop iterate, a new value of x can be non-deterministically chosen between -1 and 1.
- It is similar to a two-persons game with malicious opponent.
- What if we have a probabilistic opponent ?

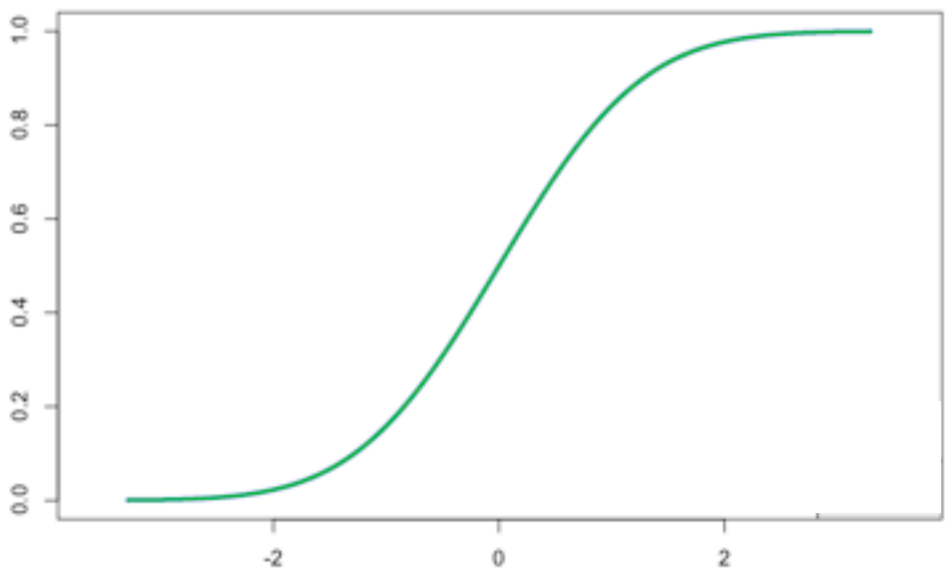
• Simulation of the program.

• Assuming inputs are independent and uniformly distributed between -1 and 1.



What is a realistic input ?

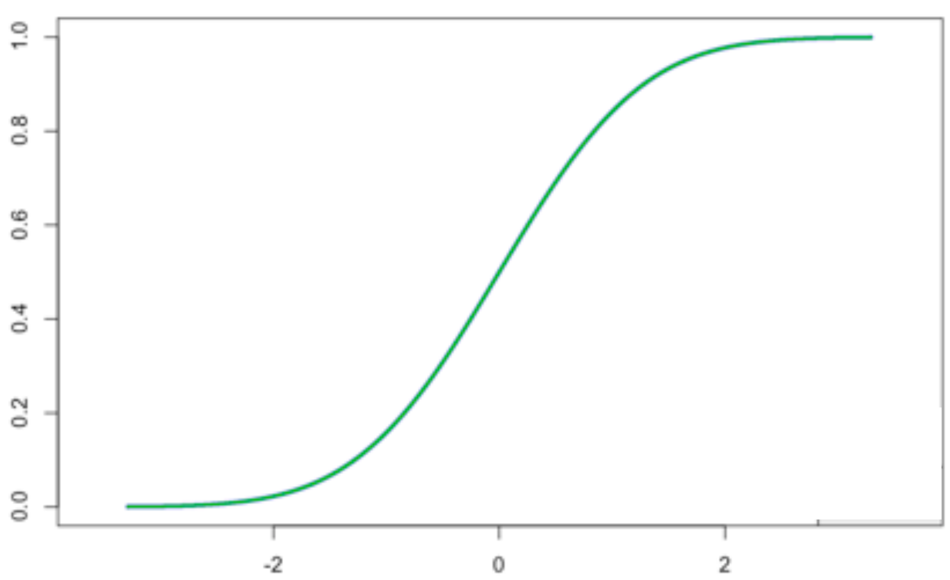
- Input values are often given by sensors that measure some physical value.
- This introduces two kinds of uncertainty:
    - non-deterministic when the evolution of the physical value is not completely determined (e.g. $\dot{y} \in [3, 3.1]$).
    - a probabilistic noise due to measurement errors.
- We need a model that can handle both kind in a uniform way.



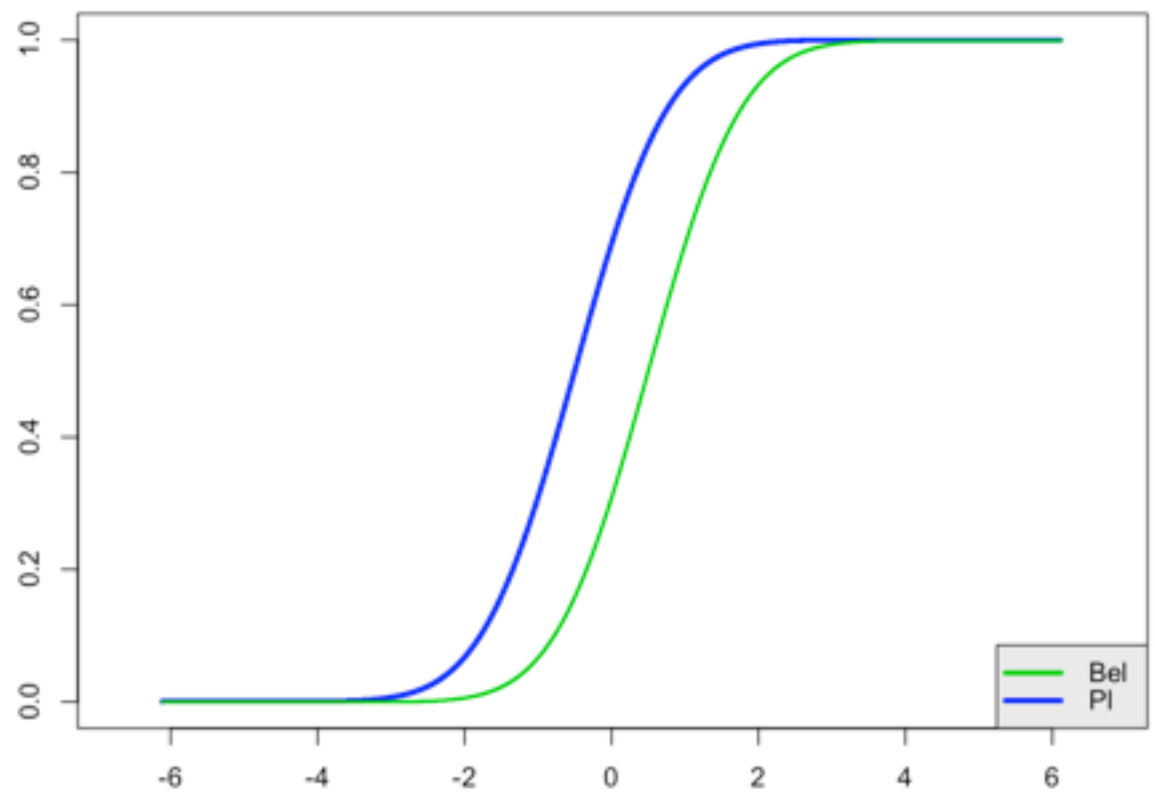
$$\boxed{\text{mean} \in [-1, 1]}$$

What is a realistic input ?

- Input values are often given by sensors that measure some physical value.
- This introduces two kinds of uncertainty:
  - non-deterministic when the evolution of the physical value is not completely determined (e.g. $\dot{y} \in [3, 3.1]$).
  - a probabilistic noise due to measurement errors.
- We need a model that can handle both kind in a uniform way.



$$\boxed{\text{mean} \in [-1, 1]}$$



Semantics of a program with probabilistic inputs.

Formally: we consider *deterministic* programs only.

- Semantics of the program.

$$[\![P]\!] : \mathbb{R}^n \to \mathbb{R}^n$$

- Adding non-determinism: collecting semantics.

$$[\![P]\!]^c : \mathcal{P}\big(\mathbb{R}^n\big) \to \mathcal{P}\big(\mathbb{R}^n\big)$$

- Abstraction: abstract semantics.

$$[\![P]\!]^\sharp : \mathbb{I}\big(\mathbb{R}^n\big) \to \mathbb{I}\big(\mathbb{R}^n\big)$$

- *Probabilistic* semantics.

$$[\![P]\!]_p : V_{\mathbb{R}^n} \to V_{\mathbb{R}^n}$$

- Adding non-determinism: *probabilistic* collecting semantics.

$$[\![P]\!]_p^c : \mathcal{P}\big(V_{\mathbb{R}^n}\big) \to \mathcal{P}\big(V_{\mathbb{R}^n}\big)$$

- Abstraction: *probabilistic* abstract semantics.

$$?$$

Semantics of a program with probabilistic inputs.

Formally: we consider *deterministic* programs only.

- Semantics of the program.

$$[\![P]\!] : \mathbb{R}^n \to \mathbb{R}^n$$

- Adding non-determinism: collecting semantics.

$$[\![P]\!]^c : \mathcal{P}\big(\mathbb{R}^n\big) \to \mathcal{P}\big(\mathbb{R}^n\big)$$

- Abstraction: abstract semantics.

$$[\![P]\!]^\sharp : \mathbb{I}\big(\mathbb{R}^n\big) \to \mathbb{I}\big(\mathbb{R}^n\big)$$

- *Probabilistic* semantics.

$$[\![P]\!]_p : V_{\mathbb{R}^n} \to V_{\mathbb{R}^n}$$

- Adding non-determinism: *probabilistic* collecting semantics.

$$[\![P]\!]_p^c : \mathcal{P}\big(V_{\mathbb{R}^n}\big) \to \mathcal{P}\big(V_{\mathbb{R}^n}\big)$$

- Abstraction: *probabilistic* abstract semantics.

?

Formally: we consider *deterministic* programs only.

- Semantics of the program.

$$[\![P]\!] : \mathbb{R}^n \to \mathbb{R}^n$$

- Adding non-determinism: collecting semantics.

$$[\![P]\!]^c : \mathcal{P}(\mathbb{R}^n) \to \mathcal{P}(\mathbb{R}^n)$$

- Abstraction: abstract semantics.

$$[\![P]\!]^\sharp : \mathbb{I}(\mathbb{R}^n) \to \mathbb{I}(\mathbb{R}^n)$$

- *Probabilistic* semantics.

$$[\![P]\!]_p : V_{\mathbb{R}^n} \to V_{\mathbb{R}^n}$$

- Adding non-determinism: *probabilistic* collecting semantics.

$$[\![P]\!]_p^c : \mathcal{P}(V_{\mathbb{R}^n}) \to \mathcal{P}(V_{\mathbb{R}^n})$$

- Abstraction: *probabilistic* abstract semantics.

?

Semantics of a program with probabilistic inputs.

Formally: we consider *deterministic* programs only.

- Semantics of the program.

$$[\![P]\!] : \mathbb{R}^n \to \mathbb{R}^n$$

- Adding non-determinism: collecting semantics.

$$[\![P]\!]^c : \mathcal{P}(\mathbb{R}^n) \to \mathcal{P}(\mathbb{R}^n)$$

- Abstraction: abstract semantics.

$$[\![P]\!]^\sharp : \mathbb{I}(\mathbb{R}^n) \to \mathbb{I}(\mathbb{R}^n)$$

- *Probabilistic* semantics.

$$[\![P]\!]_p : V_{\mathbb{R}^n} \to V_{\mathbb{R}^n}$$

- Adding non-determinism: *probabilistic* collecting semantics.

$$[\![P]\!]_p^c : \mathcal{P}(V_{\mathbb{R}^n}) \to \mathcal{P}(V_{\mathbb{R}^n})$$

- Abstraction: *probabilistic* abstract semantics.

?

Abstraction.

- We need to define abstractions of set of probability distributions.
- We need to define set-theoretic and arithmetic operations on these domains.

$$\mathcal{P}(\mathbb{R}^n)$$

$$\mathbb{I}(\mathbb{R}^n) = \{[a_1, b_1] \times \cdots \times [a_n, b_n] \mid a_i \leq b_i\}$$

$$\mathbb{Z}(\mathbb{R}^n) = \left\{ \begin{array}{l} \alpha_0^1 + \sum_{i=1}^m \alpha_i^1 \epsilon_i \\ \alpha_0^2 + \sum_{i=1}^m \alpha_i^2 \epsilon_i \\ \dots \\ \alpha_0^n + \sum_{i=1}^m \alpha_i^n \epsilon_i \end{array} \middle| \; \alpha_i^j \in \mathbb{R}, \;\; \epsilon_i \in [-1, 1] \right\}$$

$$\mathcal{P}(\mathcal{F}_{\mathbb{R}^n})$$

P-Boxes and Demster-Schafer structures.

Probabilistic affine sets.

Uncertain probabilities.

- A large community exist in the context of *uncertain probabilities*.
- It designs methods to deal with non-determinism and probability in a uniform way.
  - Mostly used in risk assessment.
  - Various terms: clouds, fuzzy sets, P-boxes, Demster-Schafer structures, ...
- Rules for combining such sets exist, but no real arithmetic is given.
- In the rest, I will present P-Boxes and D-S structures over $\mathbb{R}$.

P-Boxes: definition.

- A P-Box is given by two functions $\underline{F}$ and $\overline{F}$ such that:

$$\underline{F}, \overline{F} : \mathbb{R} \to [0,1]$$
$$\forall x \in \mathbb{R},\ \underline{F}(x) \le \overline{F}(x)$$

- $\underline{F}$ and $\overline{F}$ represent bounds on the set of distribution functions.
- When $\underline{F} = \overline{F}$, the P-Box represent *one* distribution.

- *Order*: $\left(\underline{F}, \overline{F}\right) \subseteq \left(\underline{G}, \overline{G}\right) \Leftrightarrow \forall x \in \mathbb{R},\ \underline{G}(x) \le \underline{F}(x) \le \overline{F}(x) \le \overline{G}(x)$

- *Maximal element*: $\underline{F} = \lambda x.0,\ \overline{F} = \lambda x.1$

- *Concretization*: $\gamma\left(\underline{F}, \overline{F}\right) = \left\{ P \mid \forall x,\ \underline{F}(x) \le P(X \le x) \le \overline{F}(x) \right\}$

Confidence Proof Probabilities

- Let $(\underline{F}_X, \overline{F}_X)$ and $(\underline{F}_Y, \overline{F}_Y)$ be the P-Boxes associated to variables X et Y.
- Let $Z = X \Box Y$ with $\Box \in \{+, \times, -, /\}$. How to compute $(\underline{F}_Z, \overline{F}_Z)$?
- It depends on the dependency between X and Y.
- *<u>If X and Y are independent</u>*: see DS structures.
- *<u>If X and Y are not independent</u>* (unknown dependency): Frechet bounds

$$\forall P_X \in \gamma(\underline{F}_X, \overline{F}_X),\ P_Y \in \gamma(\underline{F}_Y, \overline{F}_Y),$$

$$P(X \Box Y < u) \leq \inf_{x \Box y = u} \min\big(P_X(X < x) + P_Y(Y < y), 1\big)$$

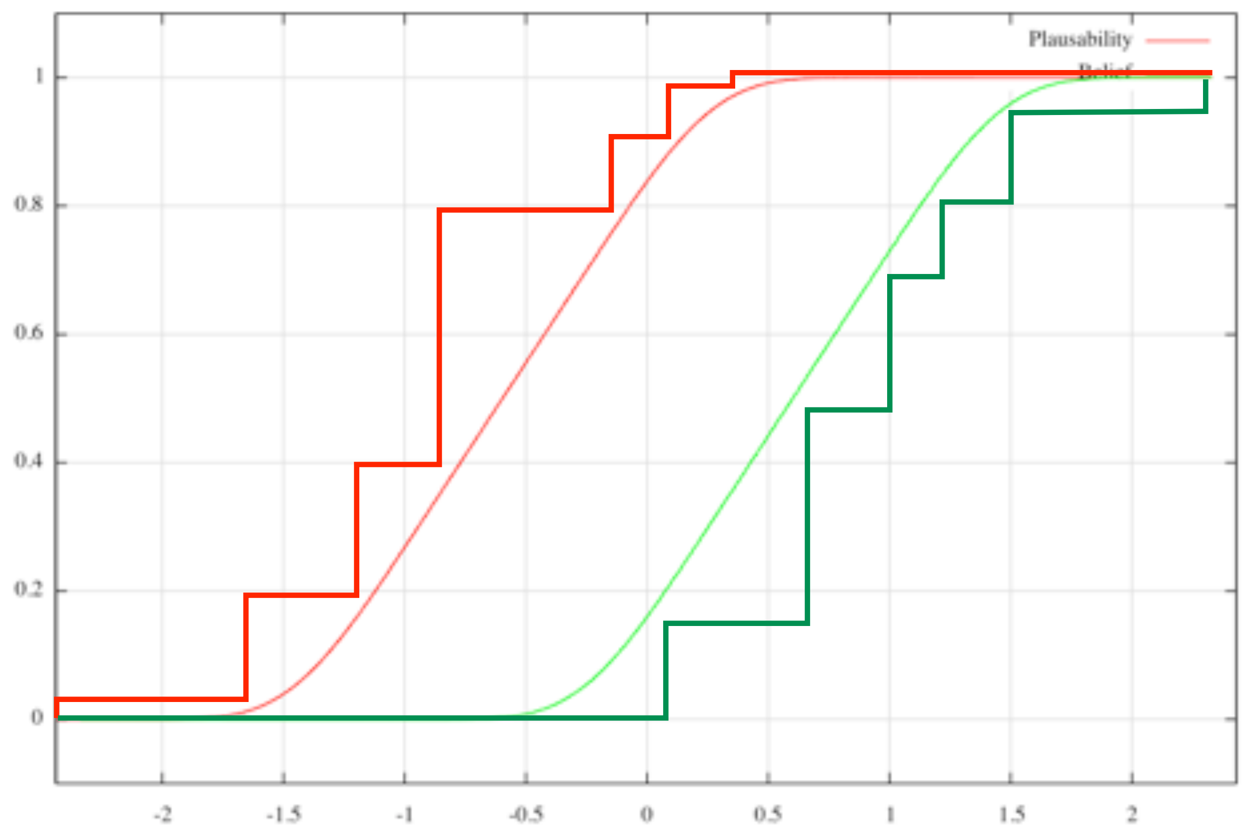
$$P(X \Box Y < u) \geq \sup_{x \Box y = u} \max\big(P_X(X < x) + P_Y(Y < y) - 1, 0\big)$$

$$\boxed{(\underline{F}_Z, \overline{F}_Z) = \lambda u. \left(\sup_{x \Box y = u} \max(\underline{F}_X(x) + \underline{F}_Y(y) - 1, 0),\ \inf_{x \Box y = u} \min(\overline{F}_X(x) + \overline{F}_Y(y), 1) \right)}$$

P-Boxes: problem of representation.

- Need to represent two continuous functions for each P-Box.
- Computation of bounds for the result of any arithmetic operator requires an optimization problem.

- One solution to solve this problem: use step functions.



- Discretization of the continuous functions.
- Introduces an over-approximation.
- Equivalent to Demster-Schafer structures.

Demster-Schafer structures: definition.

- A Dempster-Schafer structure is a set of *focal elements* associated with a probability.

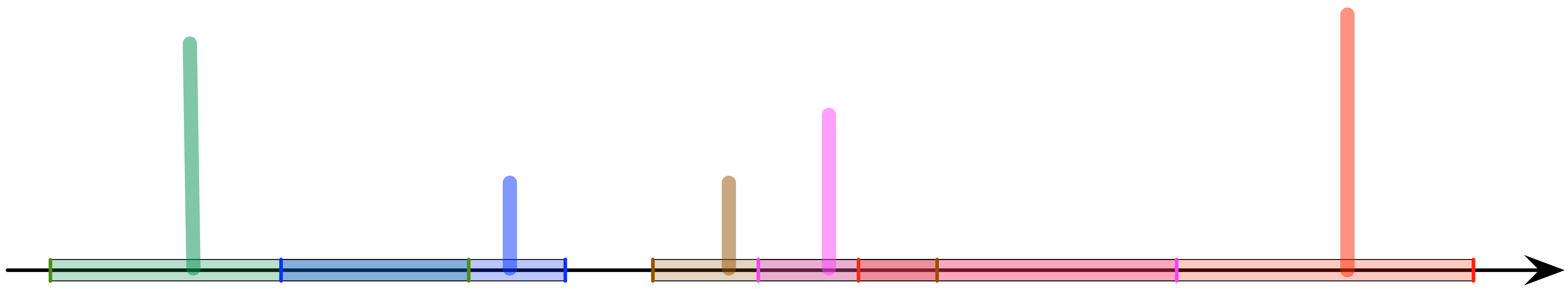
$$D = \left\{ (f_i, w_i) \mid f_i \text{ is a focal element and } w_i \in [0,1],\ \sum_i w_i = 1 \right\}$$

- Probabilistic choice between focal elements and then non-deterministic choice within the focal element.

- Usually, focal elements are *closed intervals*.

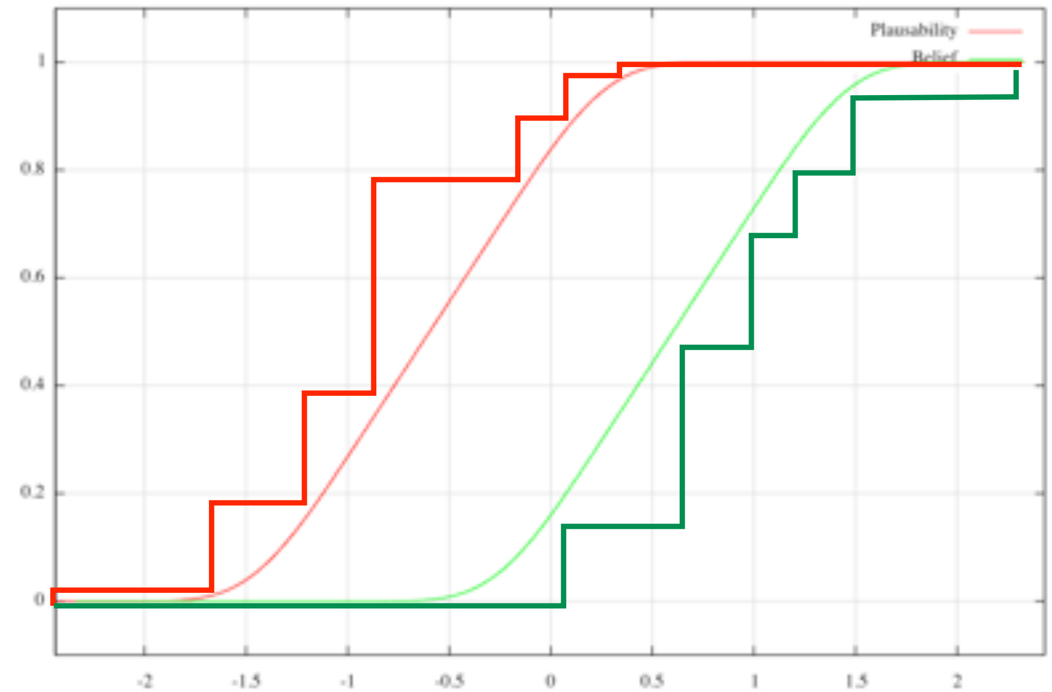
$$D = \left\{ ([a_i, b_i], w_i) \mid a_i \le b_i \text{ and } w_i \in [0,1],\ \sum_i w_i = 1 \right\}$$

- Example:

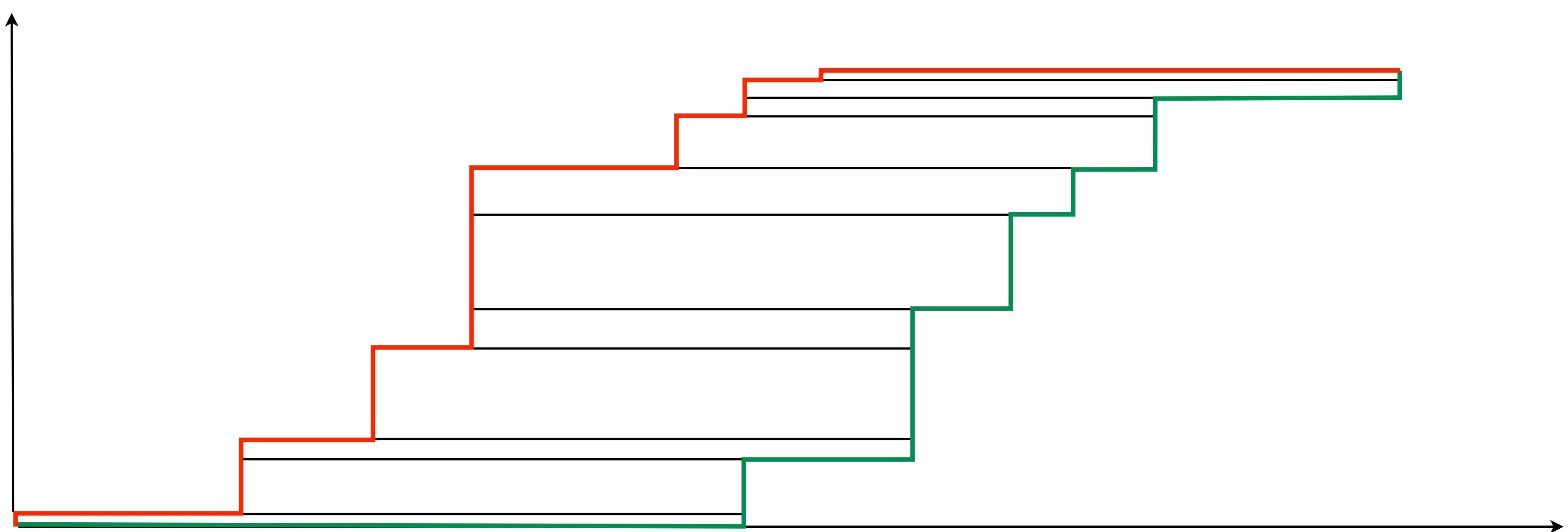


From P-Boxes to DS-structures

1. From a continuous to a discrete P-Box: discretization.



2. From a discrete P-Box to a DS-structure: Moebius inversion.



From DS-Structures to P-Boxes.

- Given a DS-Structure

$$D = \{([a_i, b_i], w_i),\ i \in [1, n]\}$$

- We define the two functions F and F by:

$$\underline{F}(x) = \sum_{i \,|\, b_i \leq x} w_i \qquad \overline{F}(x) = \sum_{i \,|\, a_i < x} w_i$$

- This forms a P-Box and

$$\gamma(D) = \gamma\left(\underline{F}, \overline{F}\right)$$

- A DS-structure thus represents a set of probability distributions enclosed by step functions.

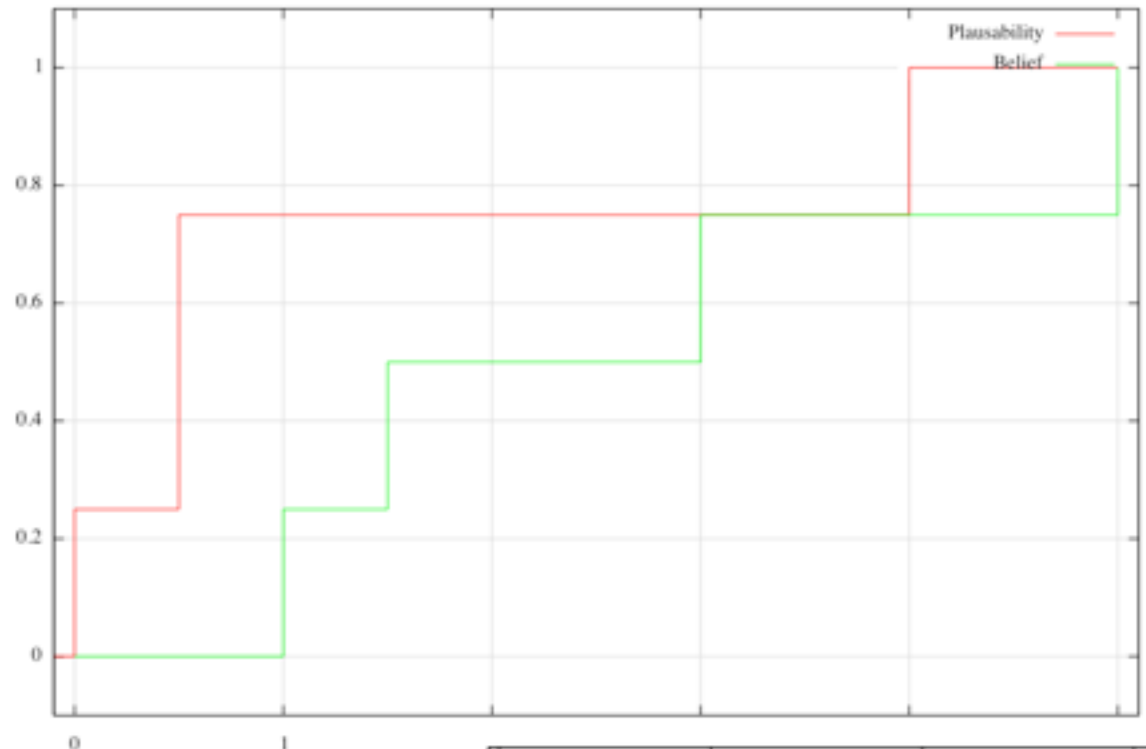
Demster-Schafer structures arithmetic.

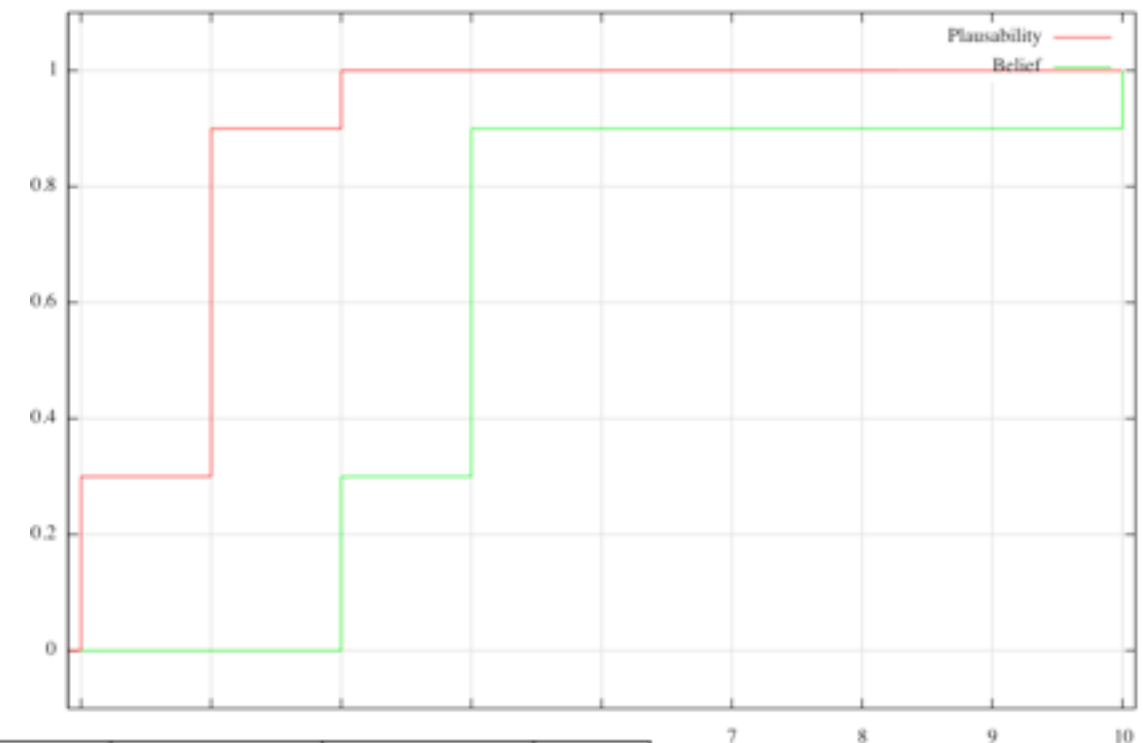
- *Case of independent inputs*.

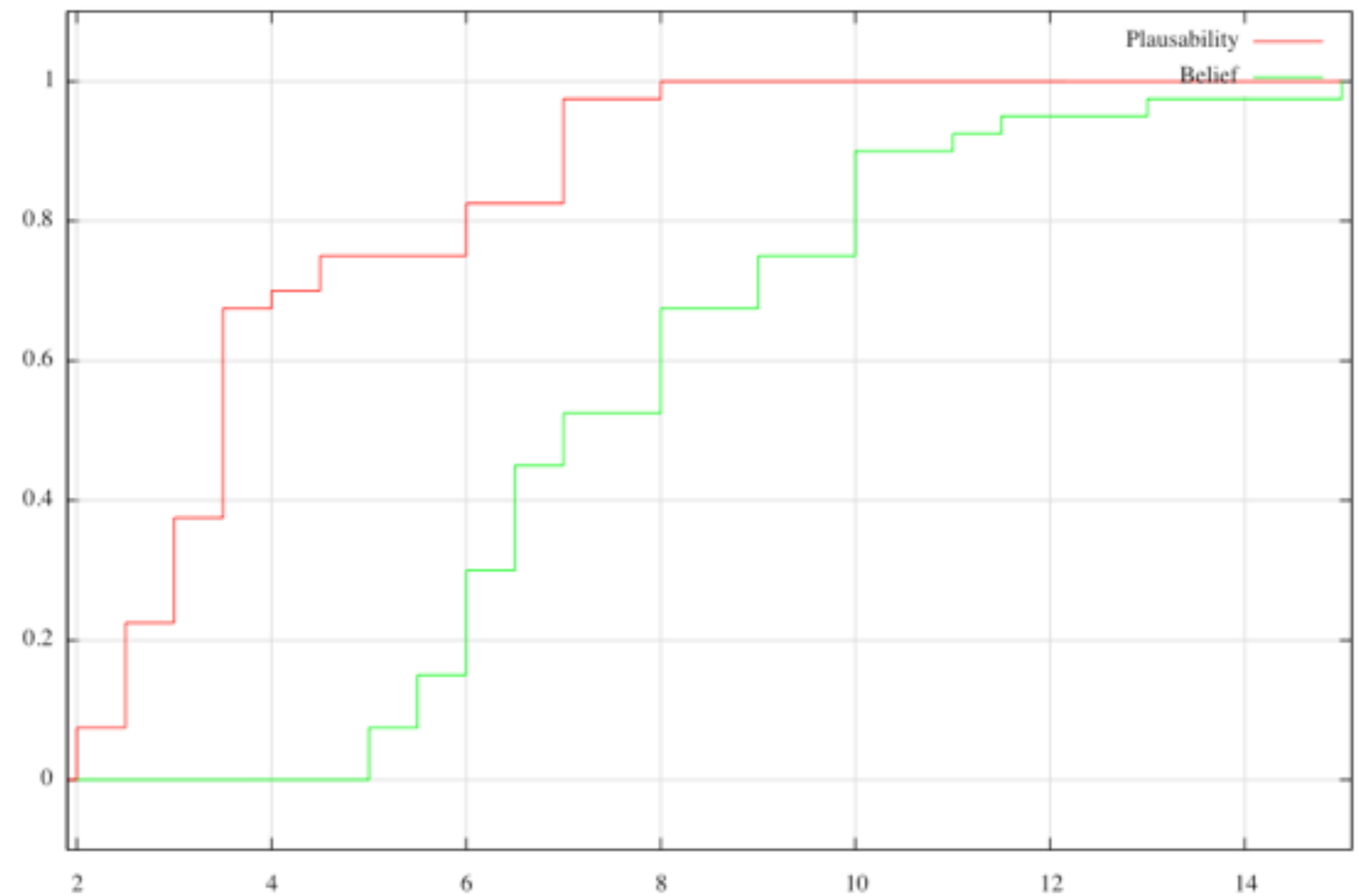
$$Z = \left\{ ([a_{ij}, b_{ij}], w_{ij}) \,|\, \exists i, j,\ [a_{ij}, b_{ij}] = [a_i, b_i] \Box [a'j, b'j],\ w_{ij} = w_i \cdot w'_j \right\}$$

Y \ X	[0,1] 0.25	[0.5,1.5] 0.25	[0.5,3] 0.25	[4,5] 0.25
[2,4] 0.3	[2,5] 0.075	[2.5,5.5] 0.075	[2.5,7] 0.075	[6,9] 0.075
[3,5] 0.6	[3,6] 0.150	[3.5,6.5] 0.150	[3.5,8] 0.150	[7,10] 0.150
[4,10] 0.1	[4,11] 0.025	[4.5,11.5] 0.025	[4.5,13] 0.025	[8,15] 0.025

Demster-Schafer structures arithmetic: example.







Demster-Schafer structures arithmetic.

- *Case of dependent inputs* (unknown dependency).

X \ Y	[2,4] 0.3	[3,5] 0.6	[4,10] 0.1
[0,1] 0.25	[2,5] p_{11}	[3,6] p_{12}	[4,11] p_{13}
[0.5,1.5] 0.25	[2.5,5.5] p_{21}	[3.5,6.5] p_{22}	[4.5,11.5] p_{23}
[0.5,3] 0.25	[2.5,7] p_{31}	[3.5,8] p_{32}	[4.5,13] p_{33}
[4,5] 0.25	[6,9] p_{41}	[7,10] p_{42}	[8,15] p_{43}

$$p_i = \sum_{j=1}^{n} p_{i,j}$$

$$p'_j = \sum_{i=1}^{m} p_{i,j}$$

$$P(Z \leq z) \leq \sum_{a'_i \leq z} p_{i,j}$$

$$P(Z \leq z) \geq \sum_{b'_i \leq z} p_{i,j}$$

Demster-Schafer structures arithmetic.

- *Case of dependent inputs* (unknown dependency).

X \ Y	[2,4] 0.3	[3,5] 0.6	[4,10] 0.1
[0,1] 0.25	[2,5] p11	[3,6] p12	[4,11] p13
[0.5,1.5] 0.25	[2.5,5.5] p21	[3.5,6.5] p22	[4.5,11.5] p23
[0.5,3] 0.25	[2.5,7] p31	[3.5,8] p32	[4.5,13] p33
[4,5] 0.25	[6,9] p41	[7,10] p42	[8,15] p43

$$p_i = \sum_{j=1}^{n} p_{i,j}$$

$$p'_j = \sum_{i=1}^{m} p_{i,j}$$

$$P(Z \leq z) \leq \sum_{a'_i \leq z} p_{i,j}$$

z=6.5

$$P(Z \leq z) \geq \sum_{b'_i \leq z} p_{i,j}$$

Demster-Schafer structures arithmetic.

- *Case of dependent inputs* (unknown dependency).

X \ Y	[2,4] 0.3	[3,5] 0.6	[4,10] 0.1
[0,1] 0.25	[2,5] p11	[3,6] p12	[4,11] p13
[0.5,1.5] 0.25	[2.5,5.5] p21	[3.5,6.5] p22	[4.5,11.5] p23
[0.5,3] 0.25	[2.5,7] p31	[3.5,8] p32	[4.5,13] p33
[4,5] 0.25	[6,9] p41	[7,10] p42	[8,15] p43

$$p_i = \sum_{j=1}^{n} p_{i,j}$$

$$p'_j = \sum_{i=1}^{m} p_{i,j}$$

$$P(Z \leq z) \leq \sum_{a'_i \leq z} p_{i,j}$$

$$P(Z \leq z) \geq \sum_{b'_i \leq z} p_{i,j}$$

z=6.5

Demster-Schafer structures arithmetic: direct method.

- *Remember* (P-Box arithmetic):

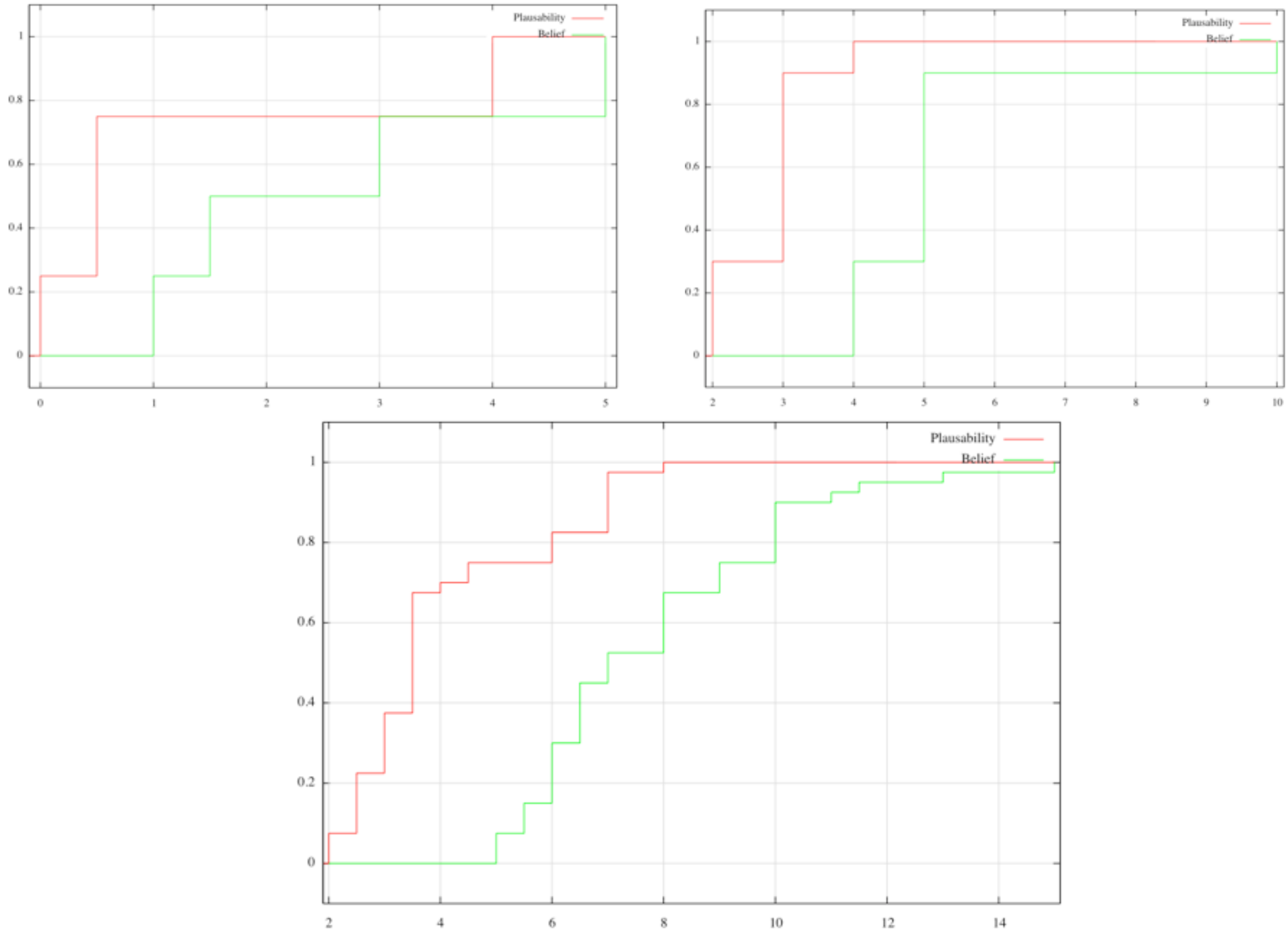
$$(\underline{F}_Z, \overline{F}_Z) = \lambda u. \Big(\sup_{x \Box y = u} \max(\underline{F}_X(x) + \underline{F}_Y(y) - 1, 0), \inf_{x \Box y = u} \min(\overline{F}_X(x) + \overline{F}_Y(y), 1) \Big)$$

- *And* (DS-structure to P-Box conversion):

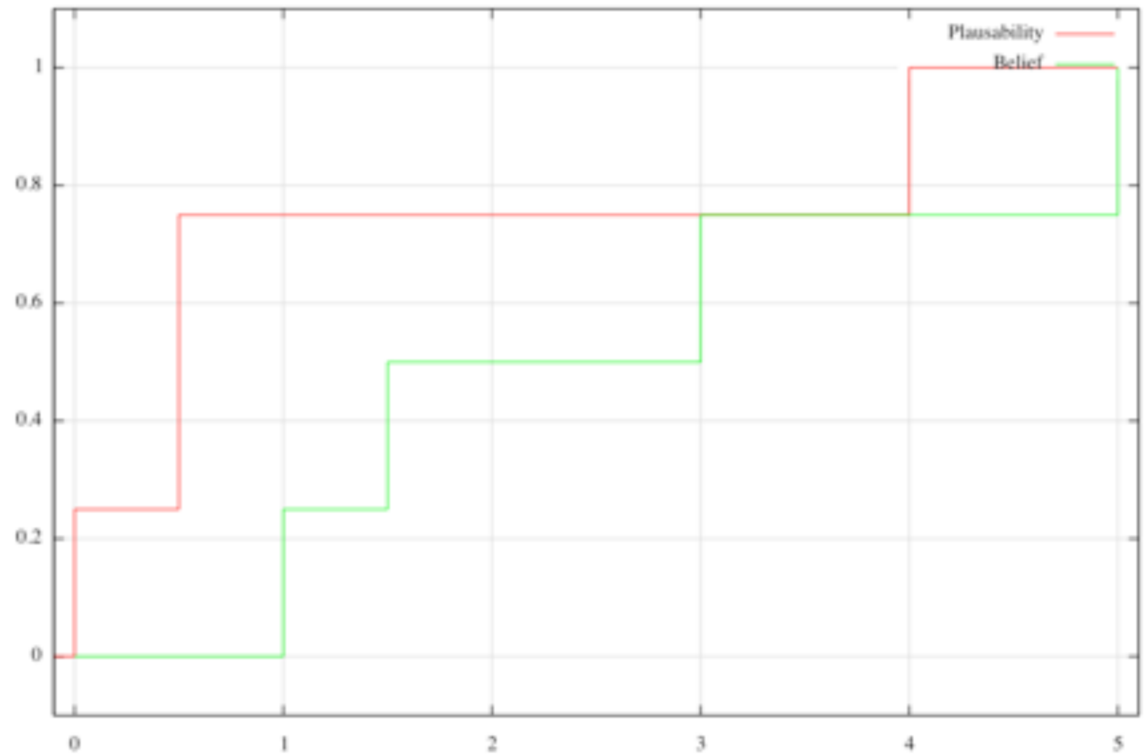
$$\underline{F}(x) = \sum_{i \,|\, b_i \leq x} w_i \qquad \overline{F}(x) = \sum_{i \,|\, a_i < x} w_i$$

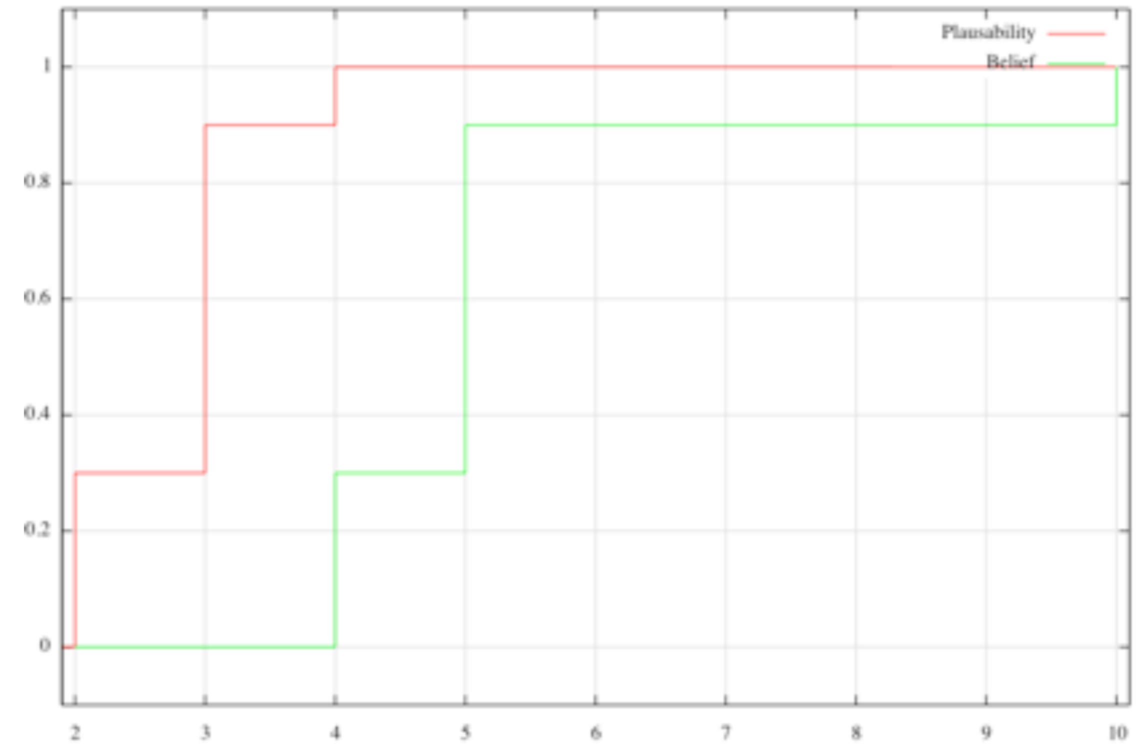
- Combining both we get a direct algorithm to compute the DS-structure of $Z = X \Box Y$ with unknown dependency.

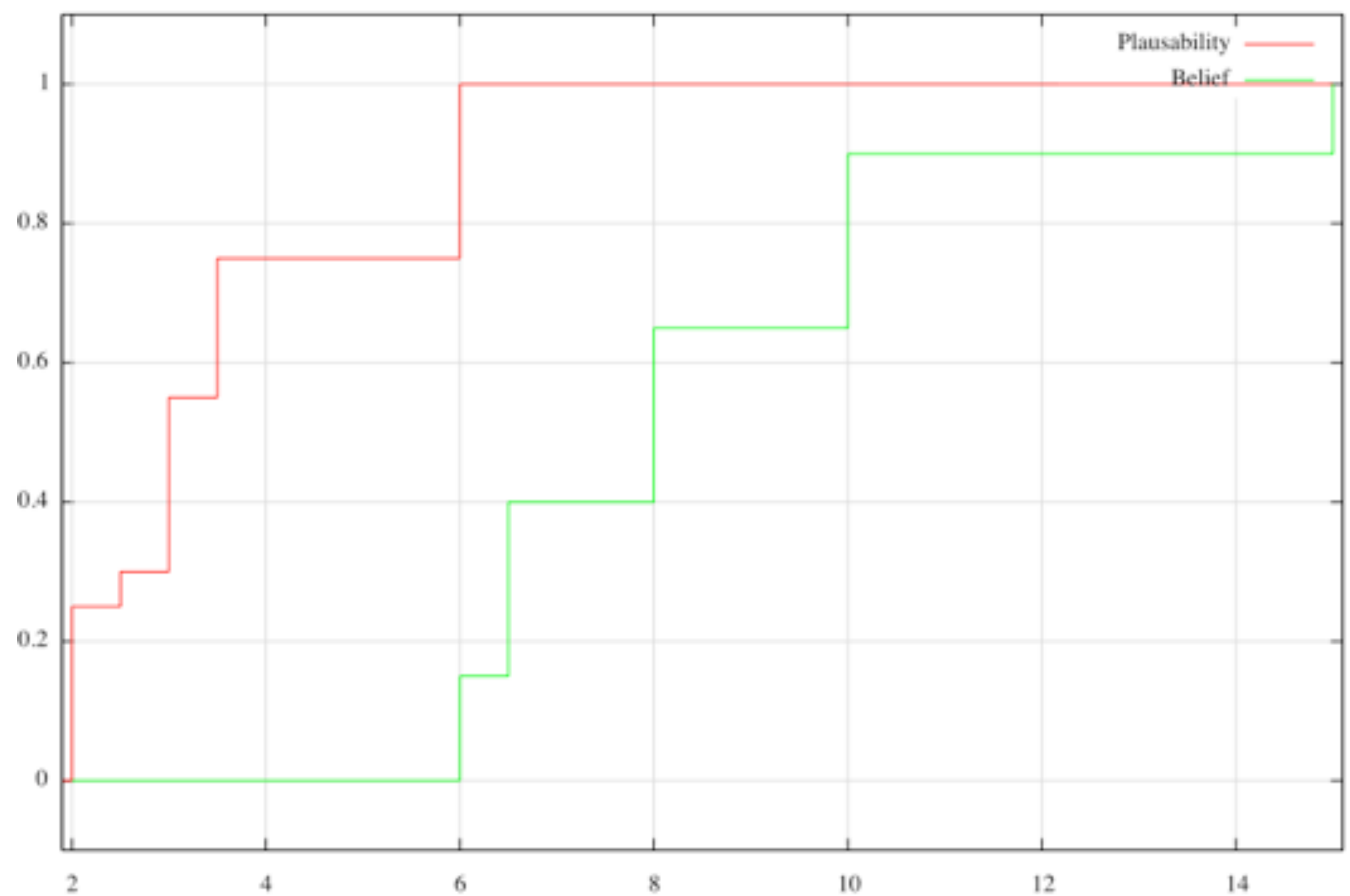
Demster-Schafer structures arithmetic: example.



Demster-Schafer structures arithmetic: example.

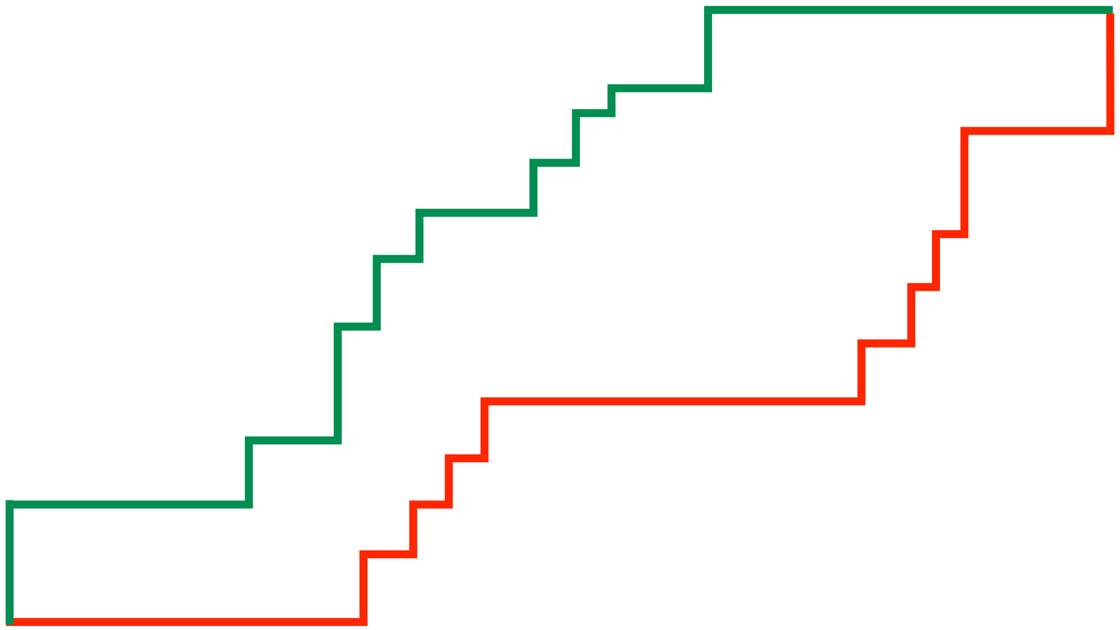






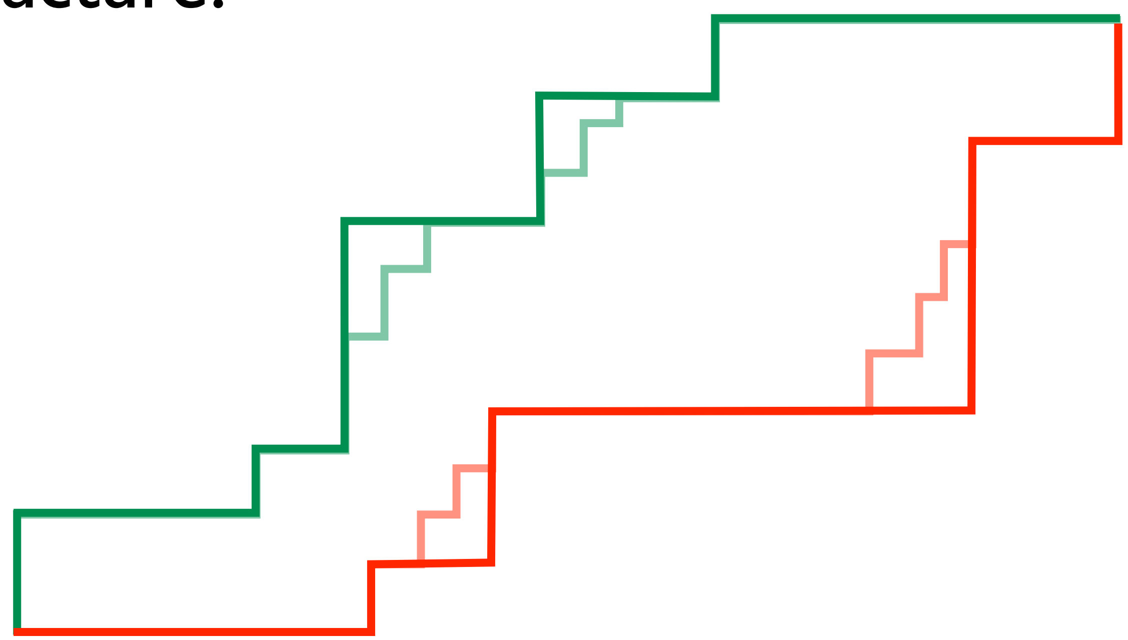
Examples of use.

- The DS-structure arithmetic has been re-implemented in C++.
- Code inspired by the Statool software.
- Implements both independent and dependent arithmetics.
- *Easy to use*: operator overloading.
- *Efficiency*: reduction operator to control the number of focal elements in the DS-structure.



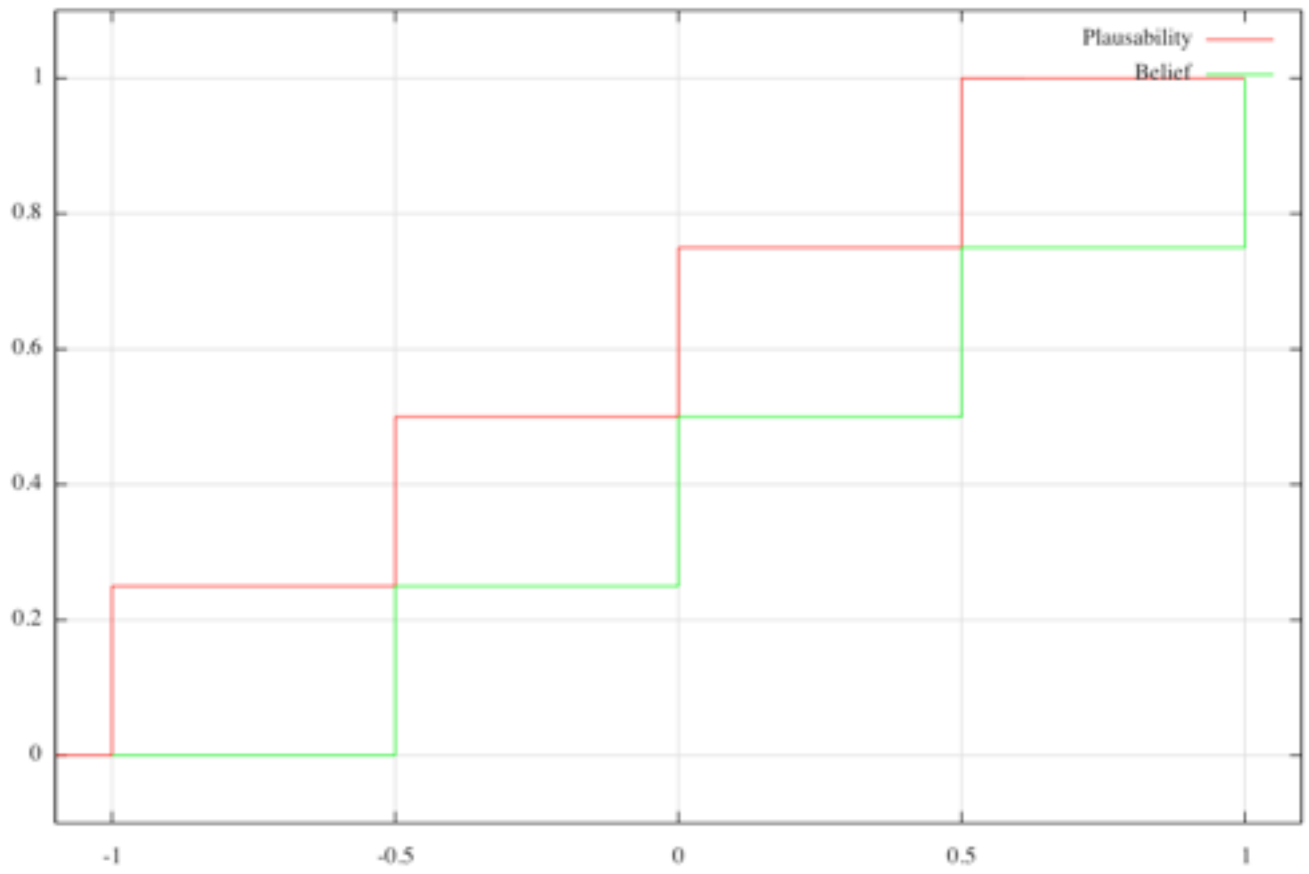
Examples of use.

- The DS-structure arithmetic has been re-implemented in C++.
- Code inspired by the Statool software.
- Implements both independent and dependent arithmetics.
- *Easy to use*: operator overloading.
- *Efficiency*: reduction operator to control the number of focal elements in the DS-structure.



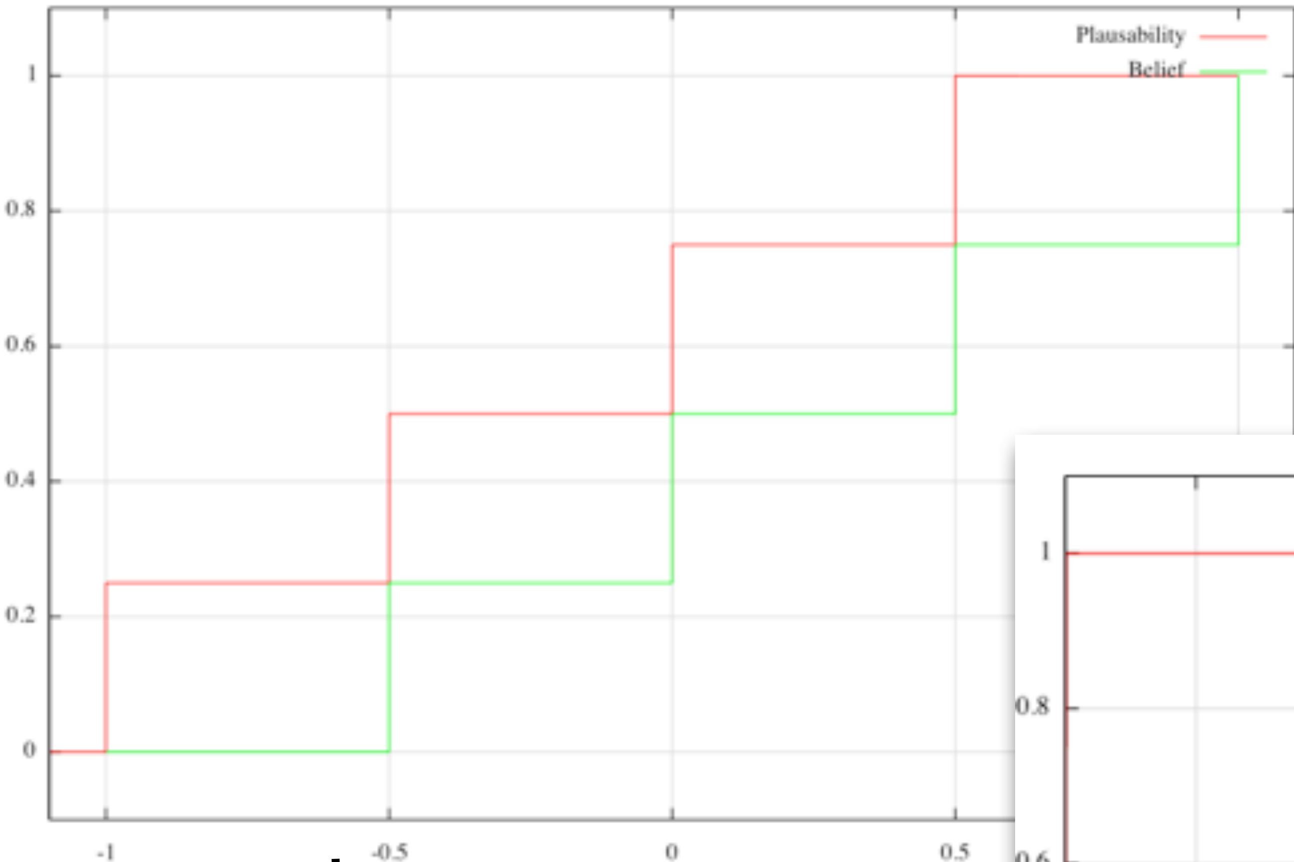
Examples of use.

- Filter with DS entries between -1 and 1.

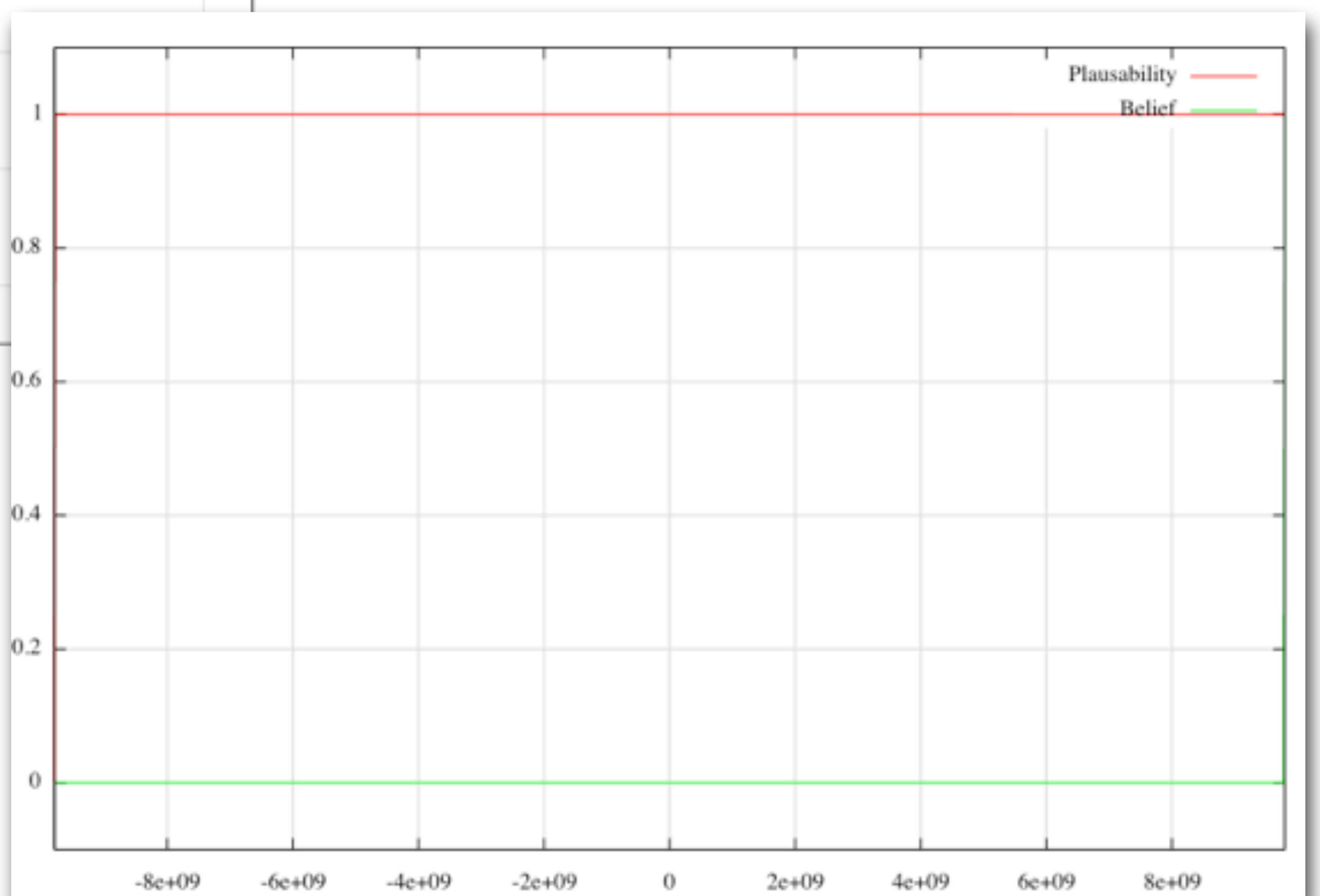


Examples of use.

- Filter with DS entries between -1 and 1.



30 iterations.



Problems with the DS-Structure approach.

- When we use the «unknown dependency» arithmetics, we make huge over-approximations.
- There are cases where the dependency between variables is known.

```
x = [-1,1];
y = [0,3];

z = x + y;
x = z + x;
```

Independent.

Known dependency.

- How can we encode this dependency ?
  - Most general way: notion of copulas.
  - Our approach: encode *linear* dependency only using affine sets.

Probabilistic affine sets.

- An affine set is the image by an affine transformation of the *n*-dimensional cube $[-1, 1]^n$
- A Demster-Schafer structure is given by a sequence of focal elements with associated probabilities.
- Two ways of mixing both:
  - either construct DS structures with focal elements being zonotopes.
  - either define the affine transformation of a DS structure with focal elements contained in the *n*-dimensional cube.
  - are both ways the same ?
- We chose the second approach and show that in this way, we introduce linear relationships between variables that help reduce the overapproximation.

Probabilistic affine sets: concept.

- A probabilistic affine set is given by:
  - an affine transformation of a set of noise symbols.
  - a function associating each noise symbol with a P-Box.

- We distinguish two kinds of noise symbols:
  - the ones representing independent inputs (ε_i). Sharing of these symbols represent linear relations between variables.
  - the ones coming from non-linear operations between variables (η_j). One such symbol is created at each non-linear operation.

- Both kinds of symbols are treated differently:
  - ε_i is supposed to be independent from $\varepsilon_{i'}$ ($i \neq i'$).
  - the dependency relation between η_j and any other noise symbol is unknown.

Probabilistic affine sets: definition.

The probabilistic affine set abstract domain

$$X_i^p = \left(X_i, \varphi^p\right)$$

$$X_i^p = \left\{ \begin{array}{ccc} x & \mapsto & \alpha_0^x + \sum_{i=1}^m \alpha_i^x \epsilon_i + \sum_{j=1}^n \beta_j^x \eta_j \\ y & \mapsto & \alpha_0^y + \sum_{i=1}^m \alpha_i^y \epsilon_i + \sum_{j=1}^n \beta_j^y \eta_j \\ & \dots & \end{array} \middle| \; \alpha_i^v \in \mathbb{R}, \; \beta_j^v \in \mathbb{R} \right\}$$

$$\varphi^p : \left\{\varepsilon_i, \eta_j\right\} \to \left\{\text{P-Boxes}\right\}$$

We call $\mathbb{AS}^p$ the set of all such elements.

Remark: we do not assume that the P-Boxes are bounded.

Probabilistic affine sets: concretization.

- The concretization function associates to a probabilistic affine set a set of probability distributions.

$$\gamma^p : \mathbb{AS}^p \to \mathcal{P}\big(\mathcal{F}_{\mathbb{R}^n}\big)$$

- Same as for afine sets, except that arithmetic operations are done using the P-Box domain.

$$\gamma^p\big(X_i, \varphi^p\big) = \Big\{ f \in \mathcal{F}_{\mathbb{R}^V} \,\big|\forall v \in V,\ f_v \in \gamma\big(\underline{F}_v, \overline{F}_v\big) \Big\}$$

$$\big(\underline{F}_v, \overline{F}_v\big) = \alpha_0^v + \sum_{i=1}^{m} \alpha_i^v \varphi^p(\varepsilon_i) + \sum_{j=1}^{n} \beta_j^v \varphi^p(\eta_j)$$

Probabilistic affine sets: concretization.

- The concretization function associates to a probabilistic affine set a set of probability distributions.

$$\gamma^p : \mathbb{AS}^p \rightarrow \mathcal{P}\left(\mathcal{F}_{\mathbb{R}^n}\right)$$

- Same as for afine sets, except that arithmetic operations are done using the P-Box domain.

$$\gamma^p\left(X_i, \varphi^p\right) = \left\{ f \in \mathcal{F}_{\mathbb{R}^V} \,\middle|\, \forall v \in V,\ f_v \in \gamma\left(\underline{F}_v, \overline{F}_v\right) \right\}$$

$$\left(\underline{F}_v, \overline{F}_v\right) = \alpha_0^v + \sum_{i=1}^{m} \alpha_i^v \varphi^p(\varepsilon_i) + \sum_{j=1}^{n} \beta_j^v \varphi^p(\eta_j)$$

Operations using independency

Operations using unknown dependency

Probabilistic affine sets: affectation.

$x = [a, b]$

$$x = \frac{a+b}{2} + \frac{b-a}{2}\varepsilon_1$$

$$\varphi^p(\varepsilon_1) = \left\{ \left\langle \left[\frac{2k-N}{N}, \frac{2(k+1)-N}{N} \right], \right\rangle \mid k \in [0, N-1] \right\}$$

Probabilistic affine sets: affectation.

- In case of non-deterministic inpus, we generate a new noise symbol, the associated P-Box being a uniform distribution.

$$x = [a, b] \qquad x = \frac{a+b}{2} + \frac{b-a}{2}\varepsilon_1$$

$$\varphi^p(\varepsilon_1) = \left\{ \left\langle \left[\frac{2k-N}{N}, \frac{2(k+1)-N}{N} \right], \right\rangle \mid k \in [0, N-1] \right\}$$

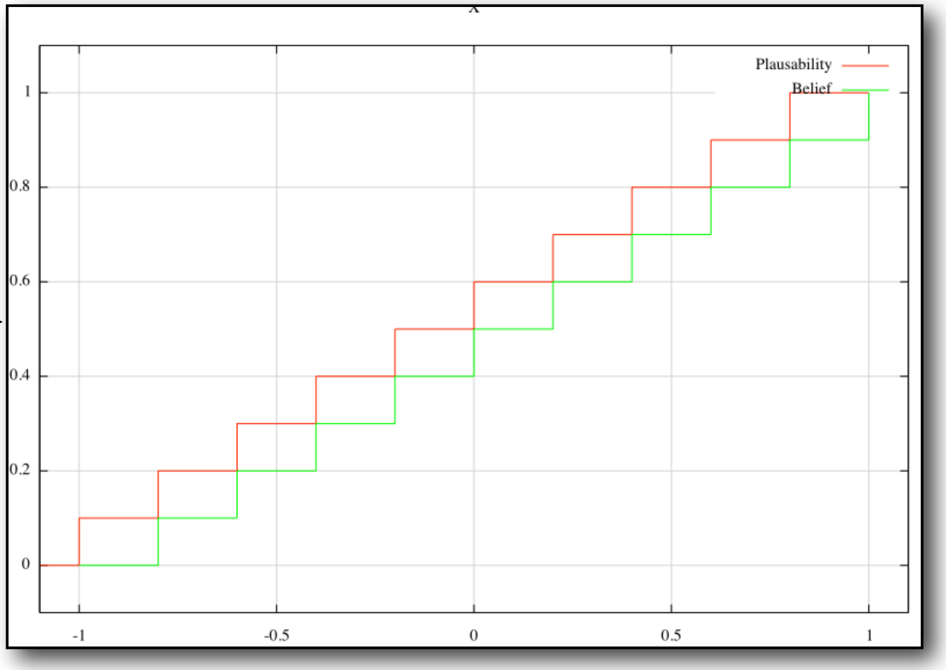
Probabilistic affine sets: affectation.

- In case of non-deterministic inpus, we generate a new noise symbol, the associated P-Box being a uniform distribution.

$$x = [a, b] \qquad x = \frac{a+b}{2} + \frac{b-a}{2}\varepsilon_1$$

$$\varphi^p(\varepsilon_1) = \left\{ \left\langle \left[\frac{2k-N}{N}, \frac{2(k+1)-N}{N} \right], \right\rangle \,|\, k \in [0, N-1] \right\}$$

$$\boxed{x = [-1, 1],\ N = 10} \longrightarrow$$

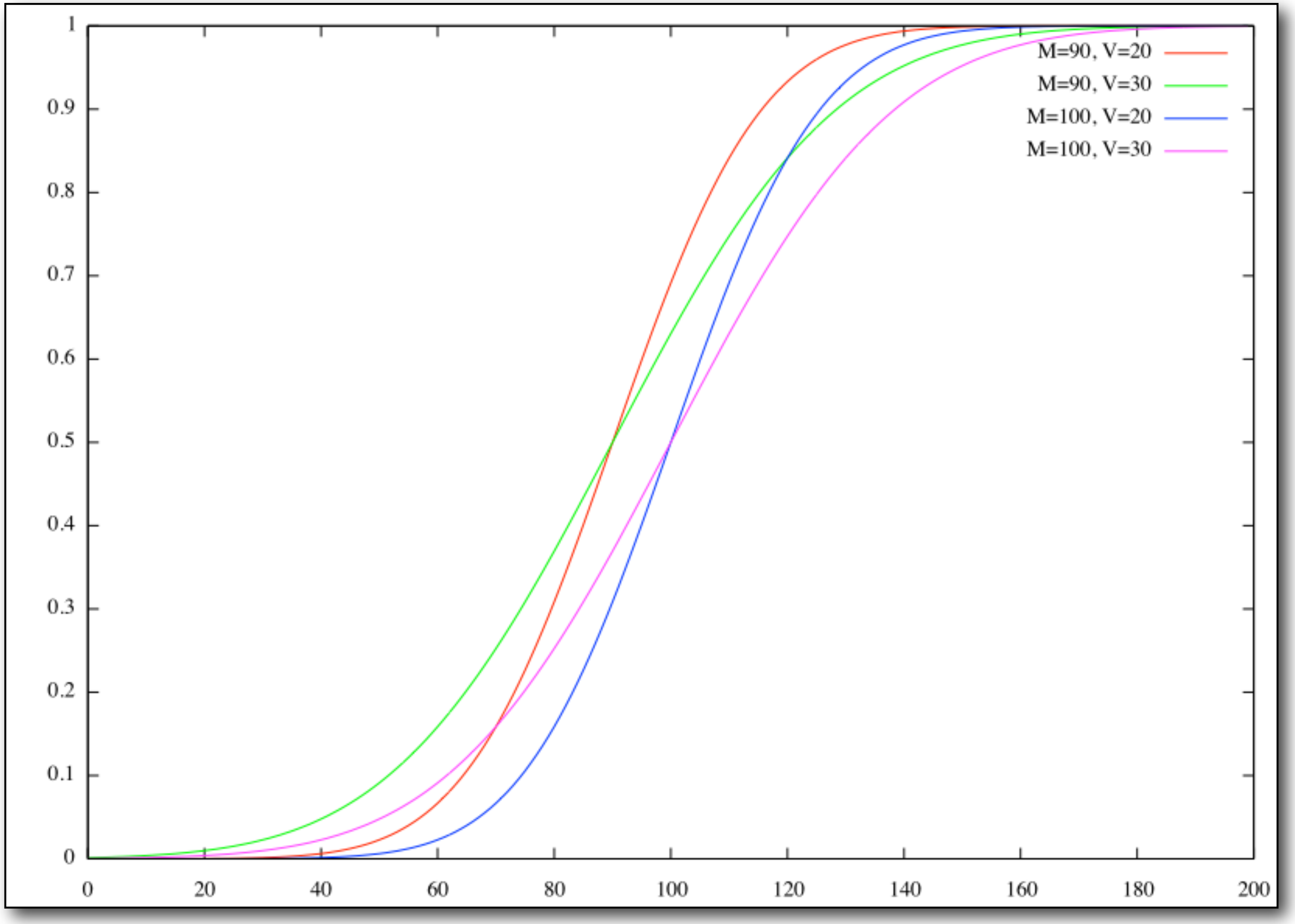


- *Remark*: even with this we may obtain «better» results than with standard affine forms, see the experimentations.

Probabilistic affine sets: affectation (bis).

- We need a more complicated affectation that can define some characteristics of the distributions (mean, standard deviation, ...).

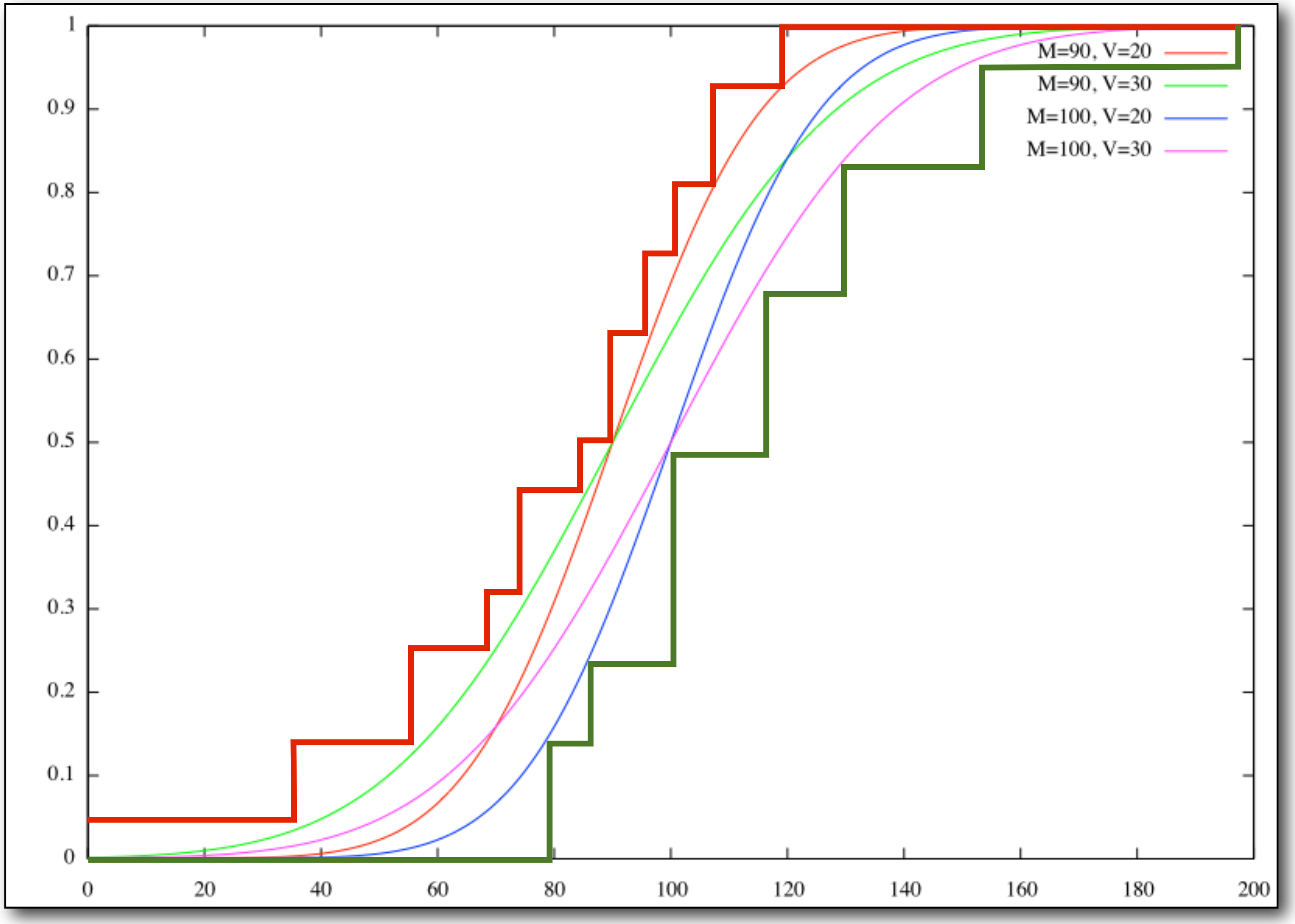
$$x = \text{normal}([90, 100], [20, 30])$$



Probabilistic affine sets: affectation (bis).

- We need a more complicated affectation that can define some characteristics of the distributions (mean, standard deviation, ...).

$$x = \mathrm{normal}([90, 100], [20, 30])$$



Probabilistic affine sets: arithmetic.

- *Linear transformation*: same as for standard affine sets.

$$f_{lin}\left(X, \varphi^p\right) = \left(f_{lin}(X), \varphi^p\right)$$

- No new symbol is created, so no need to modify φ^p.
- The dependencies between variables are propagated through f_{lin}.
- *Exemple*:

$$x = 20 - 4\epsilon_1 + 2\epsilon_3 \qquad\qquad x = 20 - 4\epsilon_1 + 2\epsilon_3$$

```
t=[1,5];
z=x-2y+t;
```

$$y = 10 - 2\epsilon_1 + \epsilon_2 \qquad \longrightarrow \qquad y = 10 - 2\epsilon_1 + \epsilon_2$$

$$z = 3 - 2\epsilon_2 + 2\epsilon_3 + 2\epsilon_4$$

$$t = 3 + 2\epsilon_4$$

$$\varphi^p \longrightarrow \varphi^p\left[\epsilon_4 \mapsto \left\{\left[\frac{2k-N}{N}, \frac{2(k+1)-N}{N}\right], \frac{1}{N}\right\}\right]$$

Probabilistic affine sets: arithmetic.

- *Multiplication*: we must define the P-Box attached to the new noise symbol.

$$
\begin{aligned}
&(\alpha_0^x + \sum_{i=1}^{m} \alpha_i^x \epsilon_i + \sum_{j=1}^{n} \beta_j^x \eta_j) \times (\alpha_0^y + \sum_{i=1}^{m} \alpha_i^y \epsilon_i + \sum_{j=1}^{n} \beta_j^y \eta_j) = \\
&\quad \alpha_0^x * \alpha_0^y + \sum_{i=1}^{m} (\alpha_0^y \alpha_i^x + \alpha_0^x \alpha_i^y) \epsilon_i + \sum_{i=1}^{m} (\alpha_0^y \eta_i^x + \alpha_0^x \eta_i^y) \eta_i \\
&\quad + \sum_{1 \le i,j \le m} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j \\
&\quad + \sum_{1 \le i \le m, 1 \le j \le n} (\alpha_i^x \beta_j^y + \alpha_i^y \beta_j^x) \epsilon_i \eta_j \\
&\quad + \sum_{1 \le i,j \le n} (\beta_i^x \beta_j^y + \beta_i^y \beta_j^x) \eta_i \eta_j
\end{aligned}
$$

- *Multiplication*: we must define the P-Box attached to the new noise symbol.

$$\left(\alpha_0^x + \sum_{i=1}^{m} \alpha_i^x \epsilon_i + \sum_{j=1}^{n} \beta_j^x \eta_j\right) \times \left(\alpha_0^y + \sum_{i=1}^{m} \alpha_i^y \epsilon_i + \sum_{j=1}^{n} \beta_j^y \eta_j\right) =$$

$$\alpha_0^x * \alpha_0^y + \sum_{i=1}^{m} (\alpha_0^y \alpha_i^x + \alpha_0^x \alpha_i^y)\epsilon_i + \sum_{i=1}^{m} (\alpha_0^y \eta_i^x + \alpha_0^x \eta_i^y)\eta_i$$

Linear part

$$+ \sum_{1 \le i,j \le m} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j$$

$$+ \sum_{1 \le i \le m, 1 \le j \le n} (\alpha_i^x \beta_j^y + \alpha_i^y \beta_j^x)\epsilon_i \eta_j$$

$$+ \sum_{1 \le i,j \le n} (\beta_i^x \beta_j^y + \beta_i^y \beta_j^x)\eta_i \eta_j$$

Probabilistic affine sets: arithmetic.

- *Multiplication*: we must define the P-Box attached to the new noise symbol.

$$(\alpha_0^x + \sum_{i=1}^m \alpha_i^x \epsilon_i + \sum_{j=1}^n \beta_j^x \eta_j) \times (\alpha_0^y + \sum_{i=1}^m \alpha_i^y \epsilon_i + \sum_{j=1}^n \beta_j^y \eta_j) =$$

$$\alpha_0^x * \alpha_0^y + \sum_{i=1}^m (\alpha_0^y \alpha_i^x + \alpha_0^x \alpha_i^y)\epsilon_i + \sum_{i=1}^m (\alpha_0^y \eta_i^x + \alpha_0^x \eta_i^y)\eta_i$$

$$+ \sum_{1 \le i,j \le m} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j$$

$$+ \sum_{1 \le i \le m, 1 \le j \le n} (\alpha_i^x \beta_j^y + \alpha_i^y \beta_j^x)\epsilon_i \eta_j$$

$$+ \sum_{1 \le i,j \le n} (\beta_i^x \beta_j^y + \beta_i^y \beta_j^x)\eta_i \eta_j$$

Non-linear part: must be overapproximated

Probabilistic affine sets: arithmetic.

- *Overapproximation of the non-linear term.*

$$\beta_{n+1}\eta_{n+1} = \sum_{1\le i,j\le m} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j + \sum_{1\le i\le m, 1\le j\le n} (\alpha_i^x \beta_j^y + \alpha_i^y \beta_j^x)\epsilon_i \eta_j + \sum_{1\le i,j\le n} (\beta_i^x \beta_j^y + \beta_i^y \beta_j^x)\eta_i \eta_j$$

- *Remark 1*: we need a normalization after that to have $\eta_{n+1} \subseteq [-1, 1]$.

- *Remark 2*: we treat separately the terms $\epsilon_i \epsilon_i$ as they are not independent.

Probabilistic affine sets: arithmetic.

- *Overapproximation of the non-linear term.*

$$\beta_{n+1}\eta_{n+1} = \boxed{\sum_{1\le i,j\le m} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j} + \sum_{1\le i\le m, 1\le j\le n} (\alpha_i^x \beta_j^y + \alpha_i^y \beta_j^x)\epsilon_i \eta_j + \sum_{1\le i,j\le n} (\beta_i^x \beta_j^y + \beta_i^y \beta_j^x)\eta_i \eta_j$$

Using independency arithmetics

- *Remark 1*: we need a normalization after that to have $\eta_{n+1} \subseteq [-1,1]$.

- *Remark 2*: we treat separately the terms $\epsilon_i \epsilon_i$ as they are not independent.

Probabilistic affine sets: arithmetic.

- *Overapproximation of the non-linear term.*

$$\beta_{n+1}\eta_{n+1} = \sum_{1\leq i,j\leq m} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j + \sum_{1\leq i\leq m, 1\leq j\leq n} (\alpha_i^x \beta_j^y + \alpha_i^y \beta_j^x)\epsilon_i \eta_j + \sum_{1\leq i,j\leq n} (\beta_i^x \beta_j^y + \beta_i^y \beta_j^x)\eta_i \eta_j$$

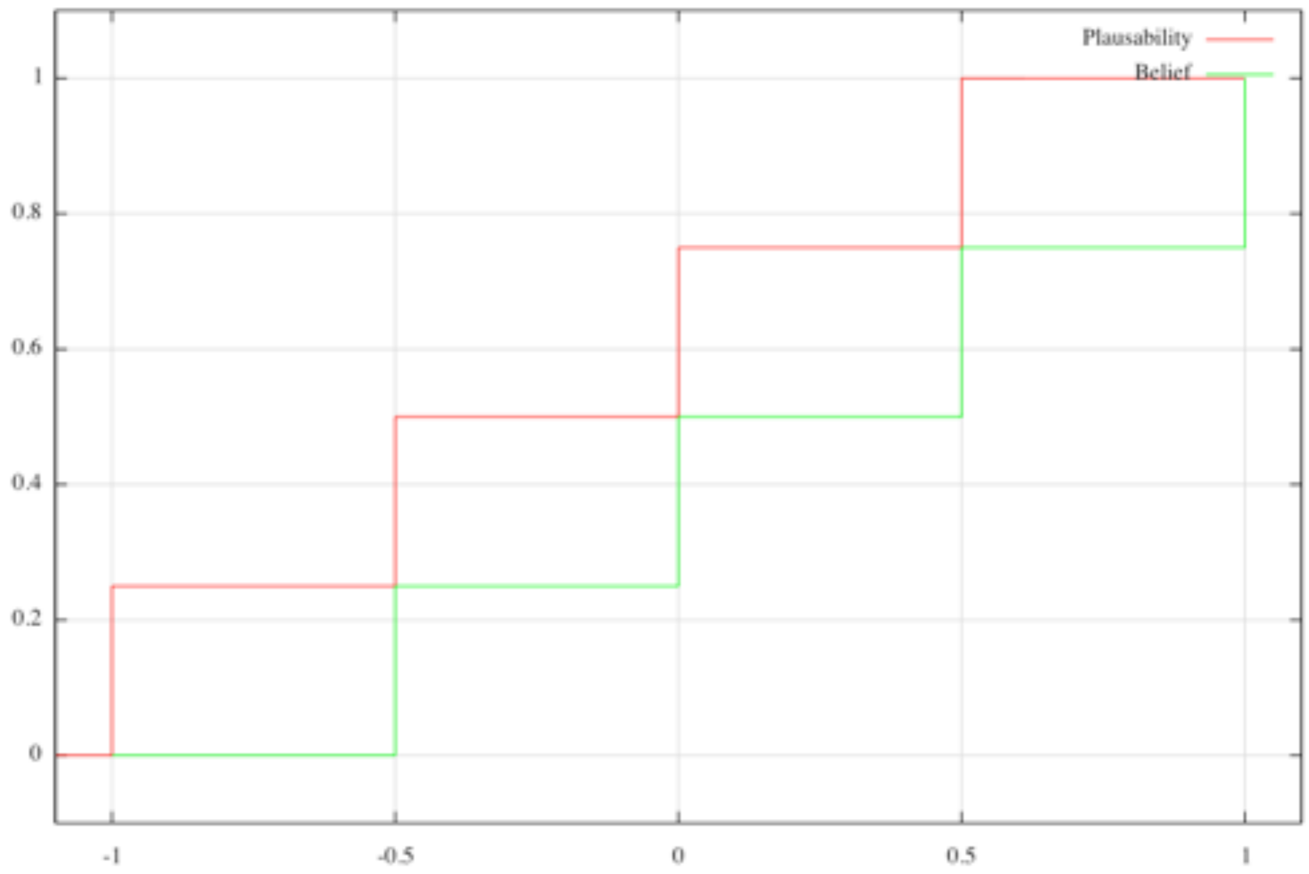
Using unknown dependency arithmetics

- *Remark 1*: we need a normalization after that to have $\eta_{n+1} \subseteq [-1,1]$.

- *Remark 2*: we treat separately the terms $\epsilon_i \epsilon_i$ as they are not independent.

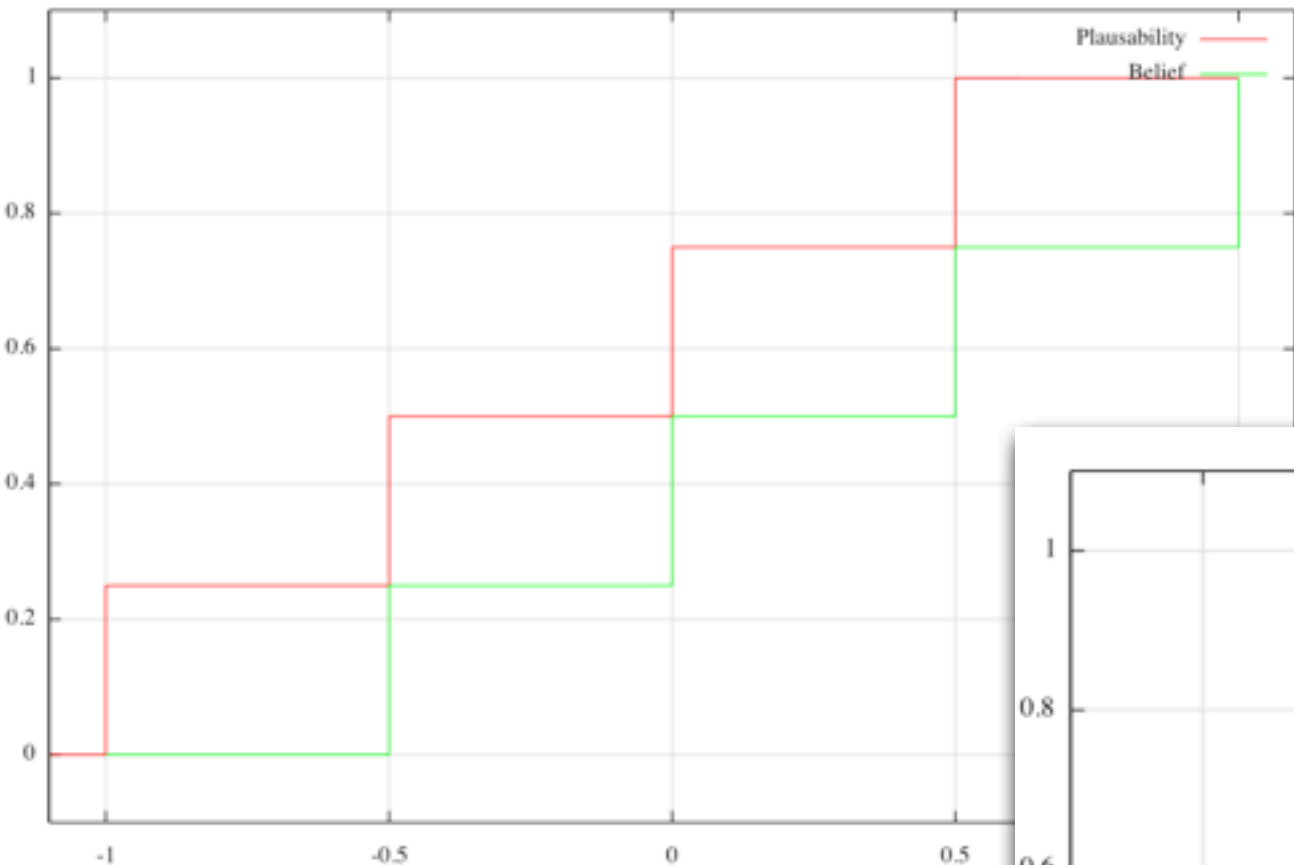
Probabilistic affine sets: example.

- Linear filter.

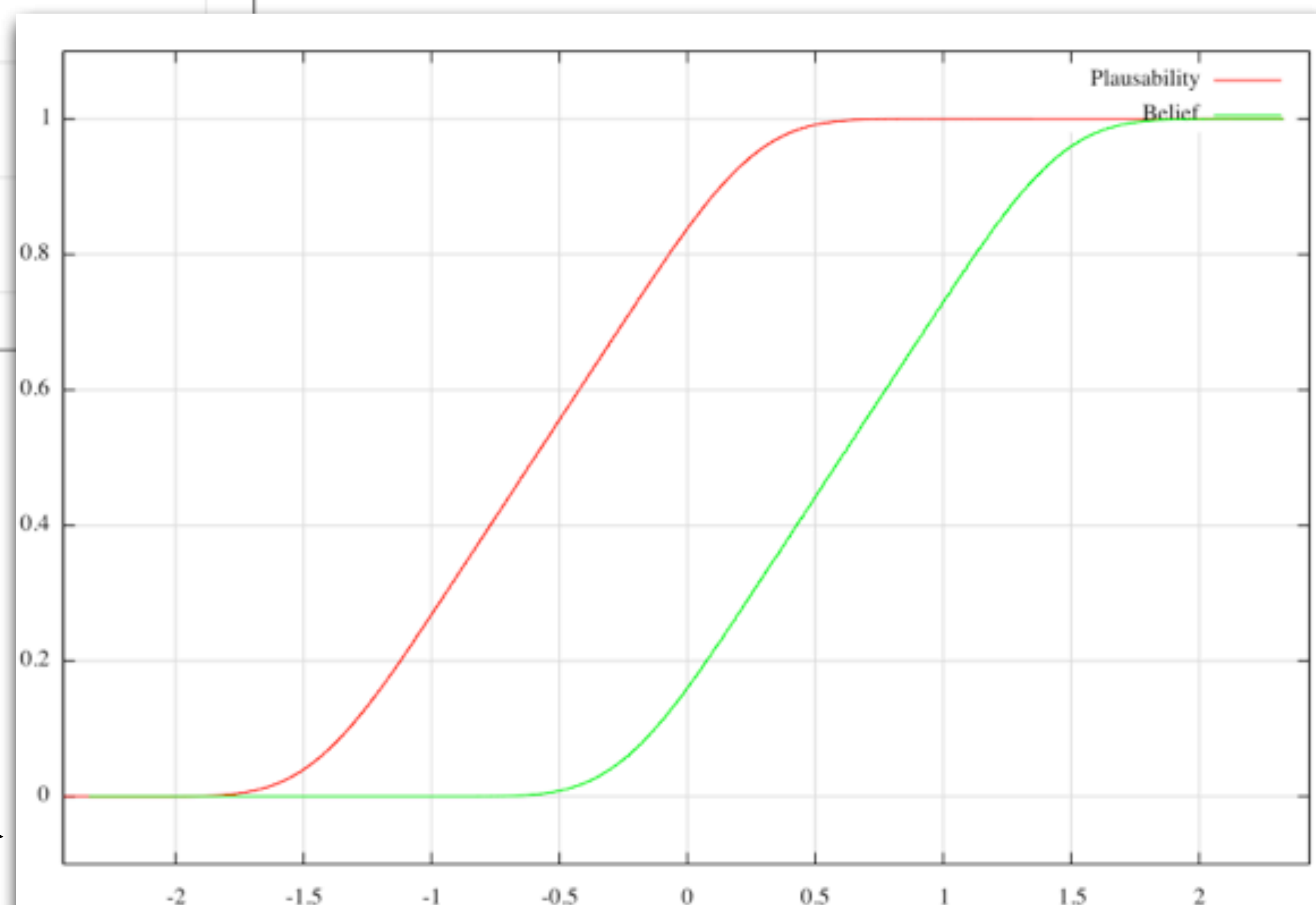


Probabilistic affine sets: example.

- Linear filter.

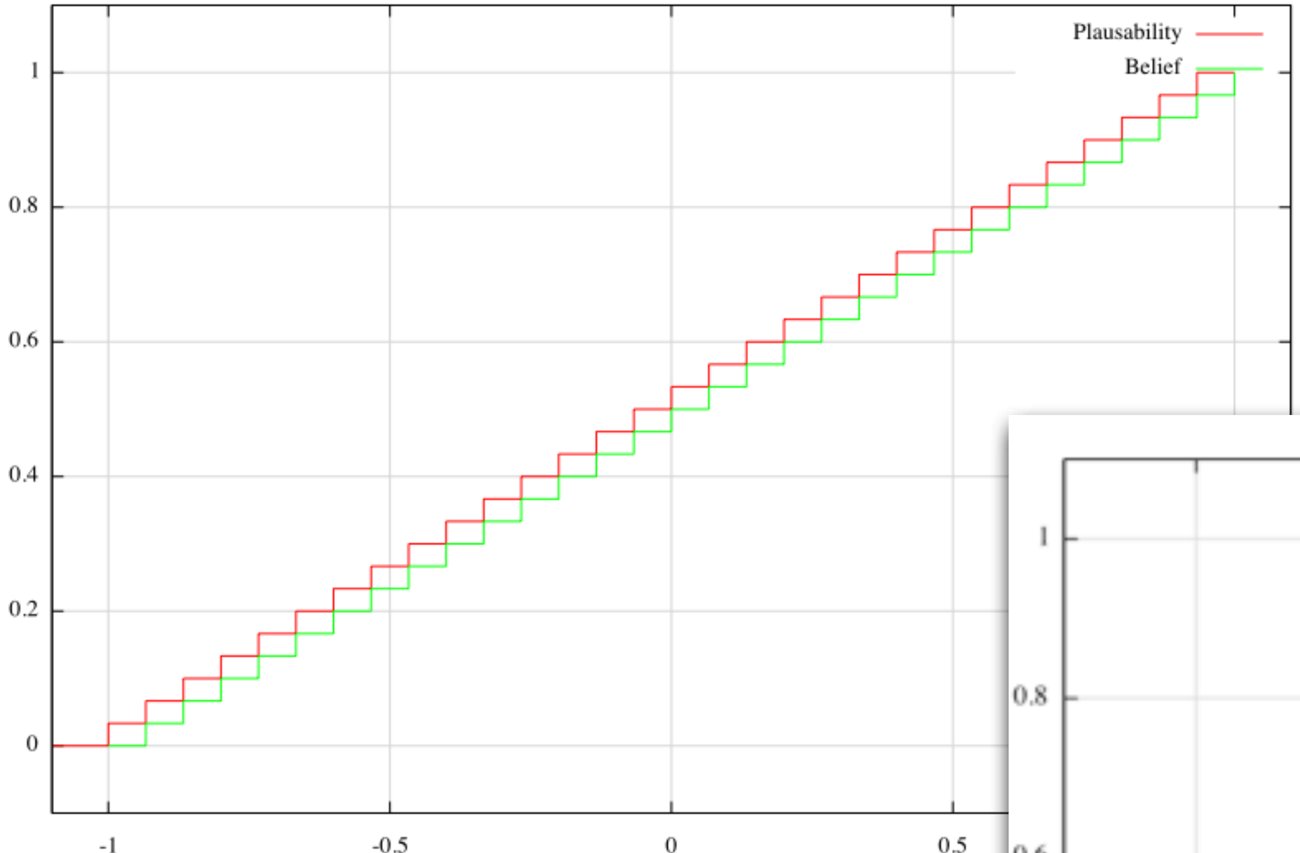


30 iterations.

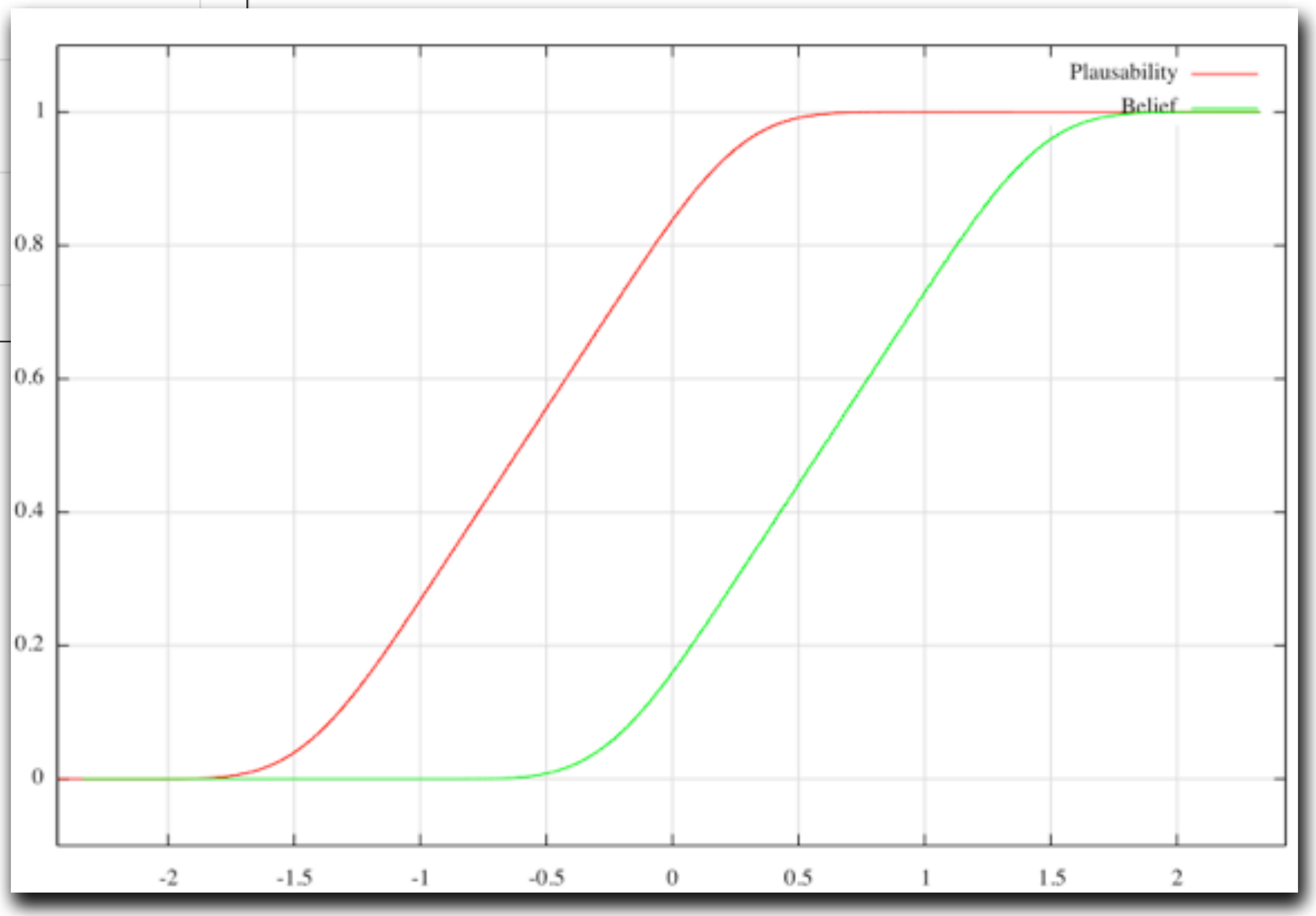


Probabilistic affine sets: example.

- Linear filter.

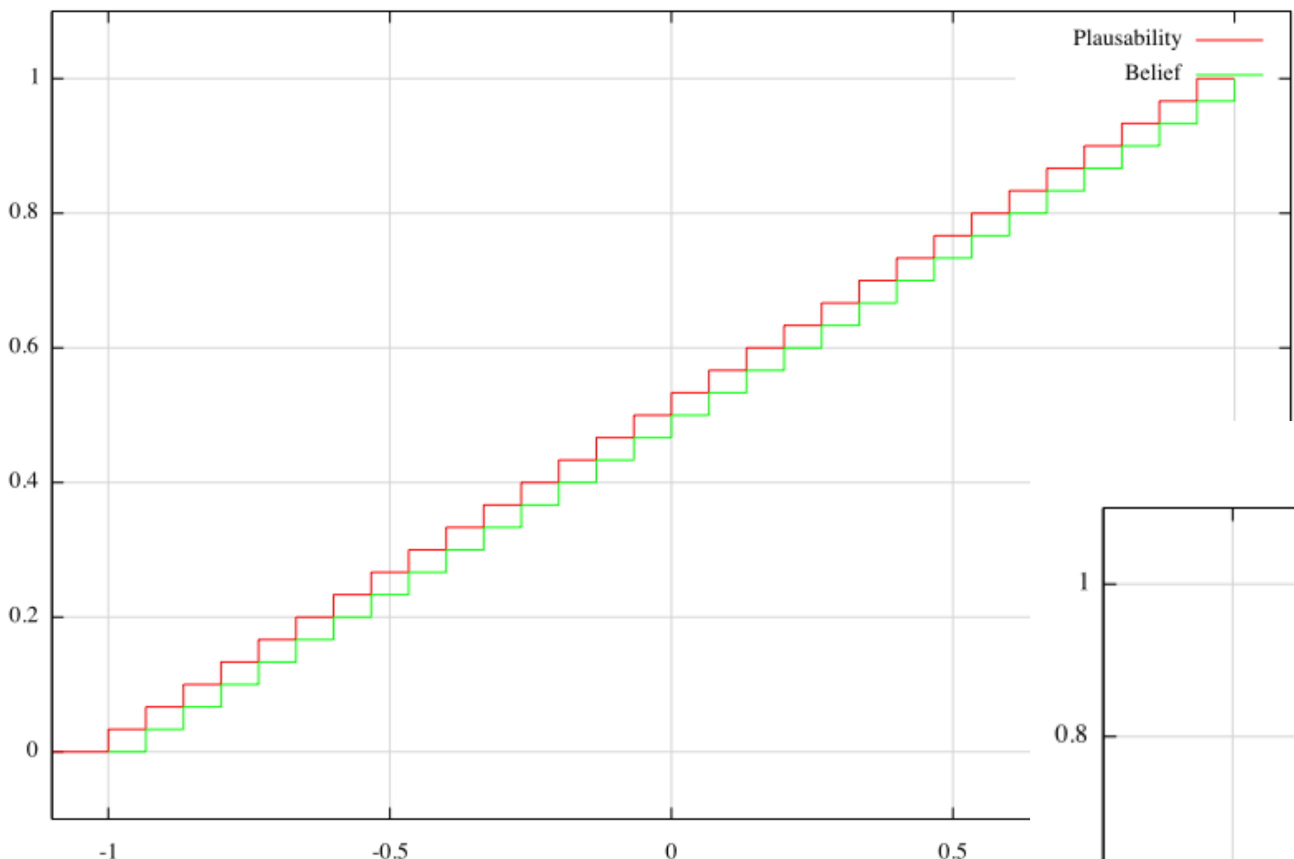


30 iterations.

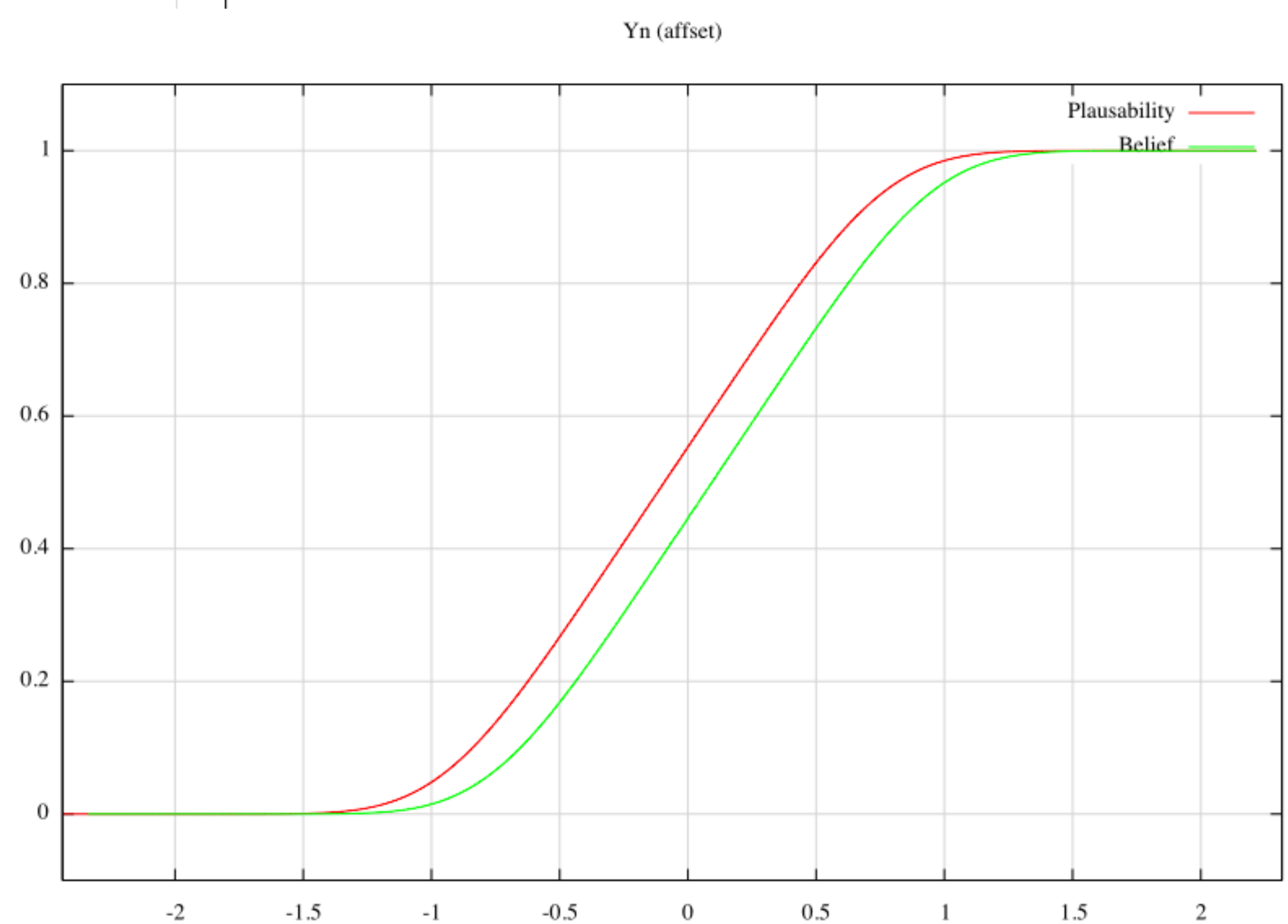


Probabilistic affine sets: example.

- Linear filter.



30 iterations.

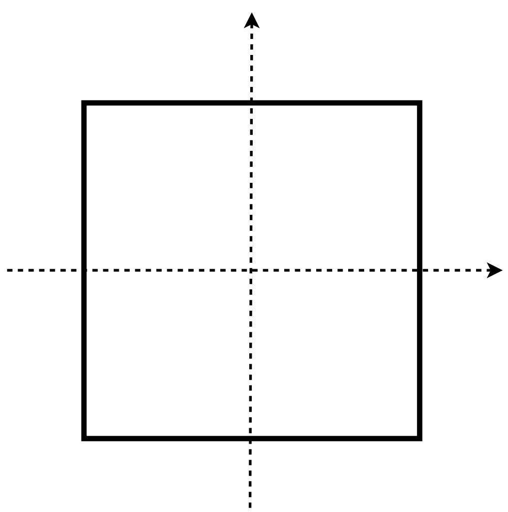


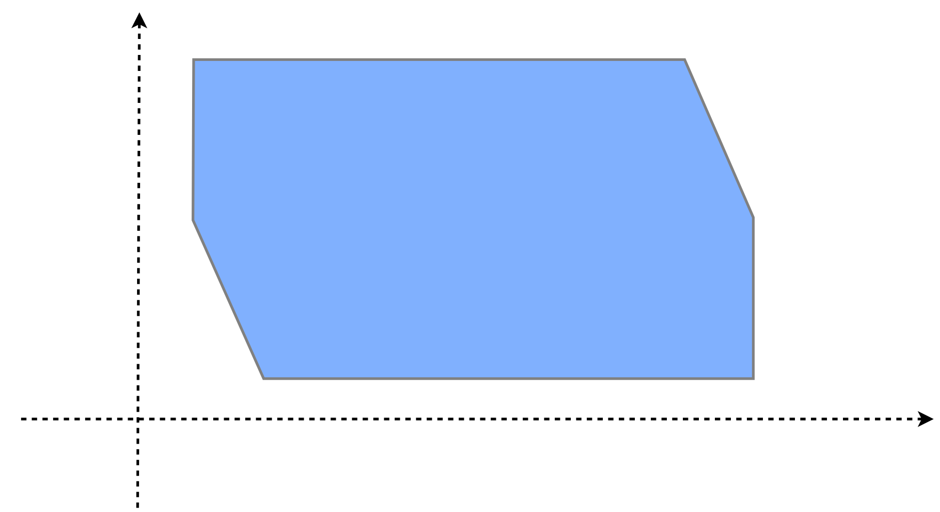
Conclusion.

Affine sets

$[-1,1]^m \xrightarrow{\text{Affine transformation.}} \mathbb{Z}(\mathbb{R}^n)$

- ✓ Arithmetic
- ✓ Order theoretic operations





$\mathbb{DS}_{[-1,1]^m} \xrightarrow{\text{Affine transformation.}} \mathbb{DS}_{\mathbb{Z}(\mathbb{R}^n)}$

- ✓ Arithmetic
- ◉ Order theoretic operations

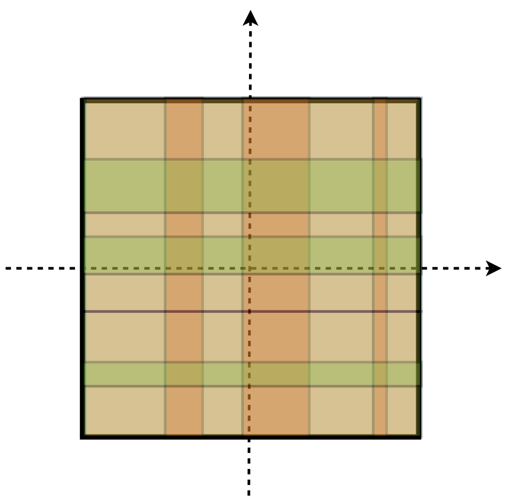
Probabilistic affine sets

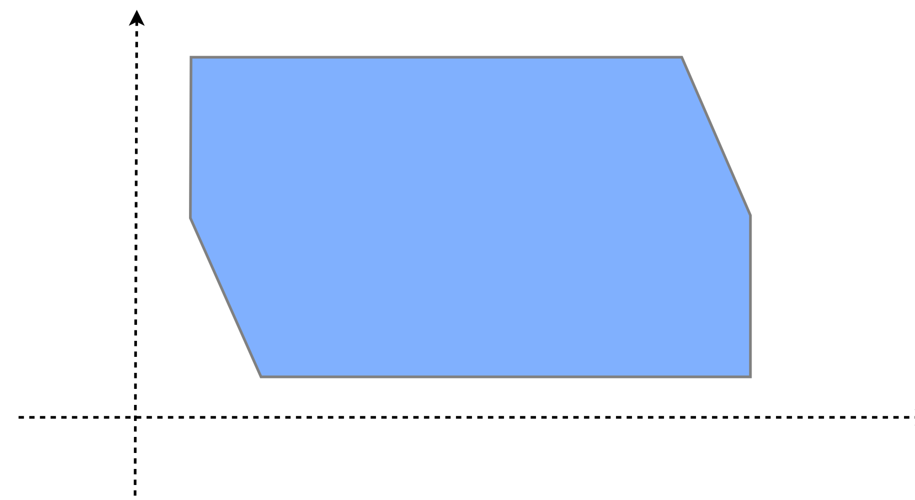
Conclusion.

Affine sets

$[-1,1]^m \xrightarrow{\text{Affine transformation.}} \mathbb{Z}(\mathbb{R}^n)$

- ✓ Arithmetic
- ✓ Order theoretic operations





$\mathbb{DS}_{[-1,1]^m} \xrightarrow{\text{Affine transformation.}} \mathbb{DS}_{\mathbb{Z}(\mathbb{R}^n)}$

- ✓ Arithmetic
- ◉ Order theoretic operations

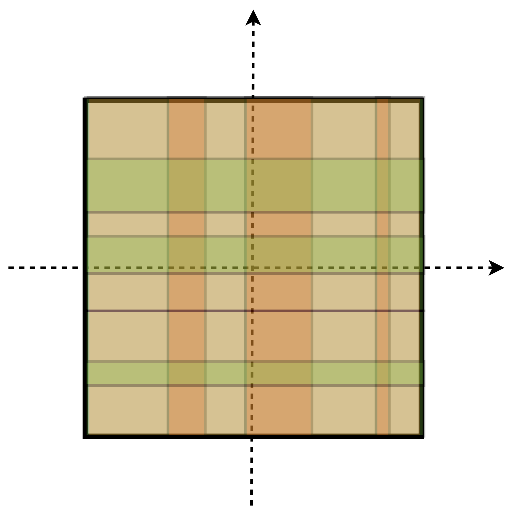
Probabilistic affine sets

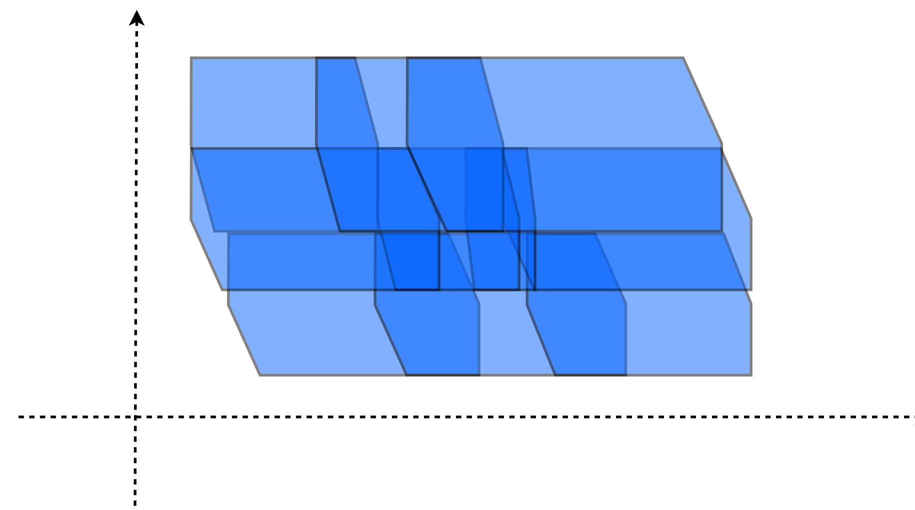
Conclusion.

Affine sets

$[-1,1]^m \xrightarrow{\text{Affine transformation.}} \mathbb{Z}(\mathbb{R}^n)$

- ✓ Arithmetic
- ✓ Order theoretic operations





$\mathbb{DS}_{[-1,1]^m} \xrightarrow{\text{Affine transformation.}} \mathbb{DS}_{\mathbb{Z}(\mathbb{R}^n)}$

- ✓ Arithmetic
- ◉ Order theoretic operations

Probabilistic affine sets

Conclusion: implementation.

- This work has been implemented in C++. See demo.