Mechanical Systems and Signal Processing 186 (2023) 109771

Contents lists available at ScienceDirect

Mechanical Systems and Signal Processing

journal homepage: www.elsevier.com/locate/ymssp

# Bivariate dependency tracking in interval arithmetic

Ander Gray [a,b,*], Marco de Angelis [b,c,*], Edoardo Patelli [b,c], Scott Ferson [b]

[a] United Kingdom Atomic Energy Authority, United Kingdom
[b] Institute for Risk and Uncertainty, University of Liverpool, United Kingdom
[c] Civil and Environmental Engineering, University of Strathclyde, United Kingdom

ARTICLE INFO

Communicated by J.E. Mottershead

Dataset link: https://github.com/AnderGray/RelationArithmetic.jl

*Keywords:*
Uncertainty propagation
Interval arithmetic
Repeated variables
Dependency tracking
Automatically verified

ABSTRACT

We propose a correlated bivariate interval arithmetic which allows for an initial dependence to be propagated, as well as the tracking of complicated non-linear dependencies arising from a computer program's execution. For this task, we extend several familiar concepts from probability theory to intervals, including bivariate copulas, conditioning, inference, and vine copulas. The interval copulas, which we call interval relations, may take any shape, and are represented by Boolean matrices defining where two intervals jointly exist or not. We use set conditioning to define an efficient correlated interval arithmetic, which may be used to find the input–output relations of operations. A key component of the presented arithmetic are interval relation networks, interval analogues to vine copulas, which store the interval relations throughout a program's execution, and use set inference to determine any unknown relations. The presented network inference can give a robust outer approximation to the exact multivariate interval dependency, which is found by projecting each pairwise bivariate relation into higher dimensions. Although some higher dimensional information is lost in this process, the bivariate projections are often sufficient to stop interval bounds becoming excessively wide. This extension allows for intervals to be rigorously and tightly propagated in deterministic engineering codes in an automatic fashion, and we apply the arithmetic on several engineering dynamics problems, including a non-linear ordinary differential equation.

## 1. Introduction

Uncertain computer arithmetic is an intrusive alternative to sampling based methods for uncertainty propagation and quantification. They rely on access to a computational model's source code, and much like how automatic differentiation calculates the derivative after every line of code, an uncertainty arithmetic calculates the uncertainty. These intrusive methods work by replacing floating point operations with operations defined for uncertain numbers, such as intervals [1–3], probability distributions [4], probability boxes [5,6] or possibility distributions [7]. This allows for uncertainty to be propagated with the following features:

- A bounded solution to the output uncertainty is achieved. That is, an outer approximation which contracts to the exact result with more computational effort. Interval calculations also allow for automatically verified computation.
- Functions of sets of probability distributions can be computed as cheaply as functions of singular distributions. For example in p-box arithmetic [6], computing with p-boxes has the same computational cost as precise distributions.

* Corresponding authors.
*E-mail addresses:* Ander.Gray@ukaea.uk (A. Gray), mda@liverpool.ac.uk (M. de Angelis), Edoardo.Patelli@strath.ac.uk (E. Patelli), Scott.Ferson@liverpool.ac.uk (S. Ferson).

https://doi.org/10.1016/j.ymssp.2022.109771
Received 21 January 2022; Received in revised form 8 August 2022; Accepted 4 September 2022

- Dependency information between variables may be partially defined. A Monte Carlo simulation requires the full input joint distribution to be specified. Any missing information about the input distribution, for example if any of the marginals or dependencies are missing, would require multiple Monte Carlo loops.

One of the reasons sampling methods are more widely used is they require no changes to source code. Preparing one's source code to be used with an uncertainty arithmetic presents its own challenges. There are two avenues for this: source-to-source transformation, and multiple dispatch. Source-to-source transformation entails some sort of a pre-compiler, where a program replaces specific data structures and functions in the source code. This would require an *uncertainty precompiler* like one described by [8], or perhaps using metaprogramming techniques. If a programming language allows it, using multiple dispatch requires by far the least effort. Multiple dispatch is a feature of a programming language which allows for multiple functions to defined with the same name. The correct function is dynamically called based on its input data type and/or other arguments. In this case only the initial variables are changed to the required uncertain types (interval, p-box, ...) and the language handles the appropriate function calls. A language like Julia [9] is very well suited for this, which has multiple dispatch as its core programming paradigm. This allows programming packages with very different functionality to work together seamlessly with very little extra work. Also in Julia custom types can be used with minimal overhead over its basic types.

The main issue with this type of uncertainty propagation is known as the *dependency problem*, or sometimes called *repeated variables* or the *wrapping effect* in interval arithmetic. When a variable appears multiple times in a sequence of operations, the same uncertainty is inserted multiple times and results are artificially inflated. By the rigorous nature of the uncertainty arithmetic, the correct solution is still bounded, however the bounds can be excessively wide. For example consider the following sequence of binary operations:

$$\begin{array}{l|l} X + Y = Z & [-1,1] + [-1,1] = [-2,2] \\ Z - X = \widetilde{Y} & [-2,2] - [-1,1] = [-3,3] \end{array} \tag{1}$$

The left shows a sequence of operations and the right shows it evaluated with interval arithmetic. $X$ and $Y$ are the intervals $[-1,1]$, and are summed to create $Z$. $X$ is then subtracted from $Z$ to create $\widetilde{Y}$. Because the *same* $X$ was added and then subtracted $Y = \widetilde{Y}$. However the calculation gave us $Y \subset \widetilde{Y}$ (i.e. $[-1,1] \subset [-3,3]$). This extra uncertainty came from the fact that $X$ was repeated in the sequence of operations. Operations between uncertain quantities are generally a function of their dependence as well as the quantities themselves. Since $Z$ was created in a binary operation involving $X$, they are somehow dependent on one another and it was the disregard of this dependence that lead to an inflated answer.

Note that what is criticised here is *naïve* or *straightforward* interval arithmetic, and not what is often called *interval computations*. In his original 1966 book, Moore [10] never proposed naïvely replacing each elementary operation with their interval equivalents as a way to estimate uncertainty. He instead suggests that, for each algorithm, we perform monotonicity checks, then use the mean value form of interval arithmetic, and perform bisection if required. For the above example, the mean value form of interval arithmetic would yield the exact interval $\widetilde{Y} = Y$. Other than the mean value form of interval arithmetic, there have been several proposed solutions for reducing artificial inflation, including significance arithmetic [11], affine arithmetic [12], zonotopes [13] and Taylor models [14]. There has also been a method proposed for probability distributions [15]. In certain cases, the expression can be rearranged such that the variable only appears once. For example realising that $a^2 + a = (a + \frac{1}{2})^2 - \frac{1}{4}$ or $\frac{a}{a+b} = \frac{1}{1+b/a}$. However this is not always applicable, for example in expressions like $x\sin(x)$, or $(x^2 + y)x$.

The fact that we can produce a rigorous enclosure of the exact range of the uncertainty is a benefit of, and perhaps unique to, interval computations. The computed range is guaranteed to include the true range of the uncertainty, so long as the input interval does. But the fact that the exact range is not always retrieved is not a fault of the method. It is known that computing the exact range of a polynomial is NP-hard [16], meaning that unless P = NP (which most computer scientists believe not to be true) no feasible algorithm can always produce the exact range. Therefore many of the set-based methods highlighted above, and indeed the method suggested in this work, aim to produce a tight enclosure of the exact range with a reasonable amount of computation.

One of the simplest-to-implement solutions is *subintervalisation*, where the interval is split into $n$ (usually linearly spaced) subintervals, and the expression is evaluated $n$ times with each subinterval. The resulting range is then taken to be the union of the propagated subintervals. An advantage of subintervalisation is that it can be used to bound the complete multivariate set of model responses, including the relationship between model inputs and outputs. For example consider a real function $f$ with $X_k$ inputs and $Y_m$ outputs, and which has been extended for intervals using interval arithmetic $f : \mathcal{B}^k \to \mathcal{B}^m$. Subintervalisation will give an outer bound to the multivariate set of input–outputs $J_{X_k Y_m}$. As an example, consider the function $f(x_1, x_2) = (y_1, y_2)$ where

$$f(x_1, x_2) = \begin{pmatrix} (x_1/(x_2^2 + 1) - 1)^2 \sin(x_2) \\ -x_1^2 + x_2^3 - x_2 \end{pmatrix}, \tag{2}$$

and with $J_{X_1 X_2} = X_1 \times X_2 = [-1.4, 1.4] \times [-1.4, 1.4]$. Subintervalisation of $J_{X_1 X_2}$ gives an outer bound on $J_{X_1 X_2 Y_1 Y_2}$, the four dimensional set of input–outputs of $f$. Three dimensional plots of this set are shown on Fig. 1. The central red shapes are three dimensional interval boxes of two of $f$'s outputs $Y_1$ and $Y_2$, and one of the inputs $X_1$ (left) or $X_2$ (right). The input set $J_{X_1 X_2}$ is not shown since it is a square. The union of the red boxes forms a rigorous bound on the set of multivariate input–outputs of $f$. If the input domain of $f$ could be subdivided to an infinite resolution, then this approach would produce the exact multivariate set of inputs and responses, known as the *united extension* [17]. The blue shapes in Fig. 1 show the two dimensional projections of the set onto the axis planes, the bivariate sets of pairwise parameters.

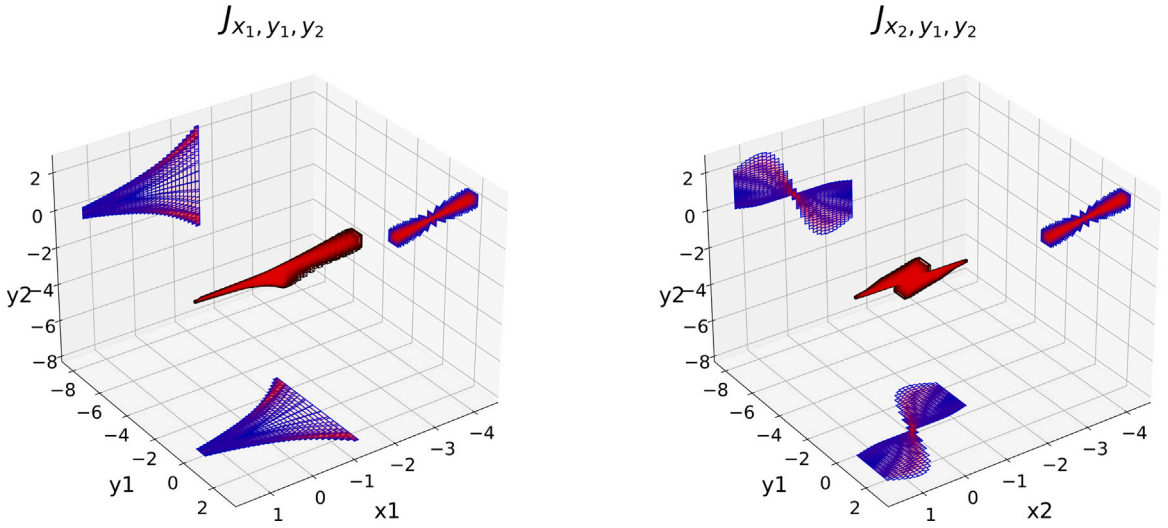


**Fig. 1.** A trivariate set of a function's input–outputs using subintervalisation.

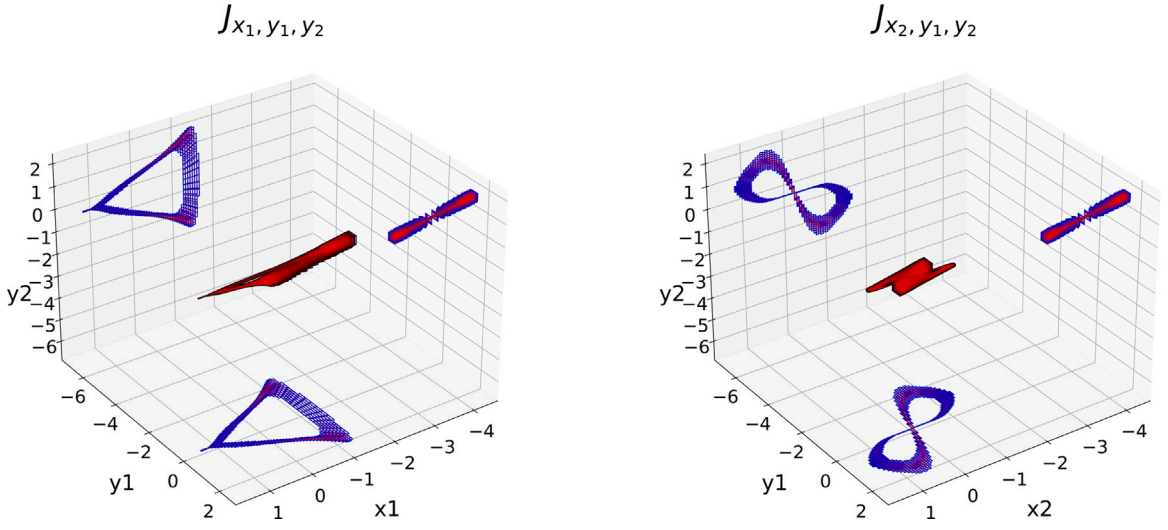


**Fig. 2.** A trivariate set of a function's input–outputs using subintervalisation, but with an initial ring shaped dependence.

An initial dependence may also be placed on the input set $J_{X_1X_2}$, such as constraining the initial set to a ring $X_1^2 + X_2^2 \subseteq [1,2]$. This may be done by first subintervalising the domain, and rejecting the boxes which do not meet the constraint. If this is performed, the set shown in Fig. 2 is obtained, which is a subset of the set shown in Fig. 1.

The drawback of subintervalisation is that it greatly suffers from the curse of dimensionality. If a function has $m$ inputs, then $n^m$ interval calculations are required for $n$ subintervals. For example a model with 9 inputs and a low subinterval number of 10 gives over 3 billion interval calculations, which may be feasible with modern high performance computing standards, but too prohibitive for standard application. Some variants of subintervalisation exist, for example interval global optimisation [18] and interval root finding [3] are based on subdividing input intervals in particular ways. Interval contractors [19] have also been used to rigorously solve inverse problems, and have been recently applied to ordinary differential equations and differential–algebraic systems of equations [20].

The approach studied in this paper considers solely the two dimensional projections of these higher dimensional sets, the blue projections shown in Figs. 1 and 2. Some of the higher dimensional information is lost when only considering projections, however the resulting dependent arithmetic is more efficient than using subintervals. We propose a model for bivariate dependencies amongst intervals, which is akin to a copula from probability theory, and show how dependent interval arithmetic may be performed, tracking dependencies in binary and unary operations. We show that complex non-convex dependencies can be tracked through linked operations using this method.

In functions involving multiple variables, we propose an interval relation network model to store the known or calculated binary dependencies, and to infer the binary dependencies required in the course of a calculation. This network model is similar to a vine copula from probability theory, and may be used to bound higher dimensional sets from the known binary projections in a rigorous way.

The presented method preserves the rigour of interval arithmetic, and a varying granularity or resolution may be defined, with a higher resolution providing tighter answers but a higher computational cost. Parallelisation of this method is however straightforward, with some discussions provided.

## 2. Dependence amongst intervals

There have been a number of proposed models for dependencies amongst intervals. Ferson and Kreinovich [21] describe a dependence between two intervals as regions where the two sets are pairwise allowed and disallowed (values where the two sets jointly exist). They describe a number of parametric families for interval dependencies, and show how arithmetic operations can be performed with these families. They include *non-interactive* (the standard interval arithmetic case), *perfect* and *opposite* dependencies as special cases. Ceberio et al. [22] similarly describe a joint interval as a set of possible allowed pairs, but allow the set to take any shape. They define a dependence between two intervals $X_1$ and $X_2$ to be any proper subset of their Cartesian product $J_{X_1X_2} \subseteq X_1 \times X_2$. They describe how any arbitrary subset of $X_1 \times X_2$ can be represented on a computer by a discrete *outer* approximation. They first divide the box $X_1 \times X_2$ into $n \times n$ subboxes; and then describe the set $J_{X_1X_2}$ as the union of subboxes which contain a possible pair. This representation of a set is the well known upper approximation used in *rough set* theory [23]. They perform dependent interval operations with interval arithmetic on each individual subbox, much in the style of subintervalisation. As opposed to storing each subinterval, which would require $n^2 \times 2$ (one for each interval bound) floating point numbers, they propose that the sets are represented by Boolean matrices, requiring only $n^2$ bits. In this section we expand on the ideas of Ceberio et al. [22], describing how bivariate intervals may be defined and manipulated. We also describe a method for performing an arithmetic with dependent intervals, which is computationally cheaper than subintervalisation and may be used to determine dependencies of inputs and outputs of binary and unary operations, a task required for dependency tracking.

### 2.1. Interval relations

In probability theory, a bivariate copula returns the value of the cumulative distribution for a $(u, v)$ pair in the unit square $[0,1]^2$, $C : [0,1]^2 \to [0,1]$, and can be used to define any probabilistic dependency independently from marginals. Following a similar idea, we define any generic interval dependency as an indicator function $\mathcal{R}(u, v)$ on $[0,1]^2$ which returns a Boolean value, $\mathcal{R} : [0,1]^2 \to \{0,1\}$, stating whether the pair $(u, v)$ is contained in the bivariate set. Much like how univariate intervals make no statement of how the probability is distributed within them, these bivariate sets only state whether possible pairs are contained or not. Generally the set $\mathcal{R}$ may be any arbitrarily complicated two dimensional shape, which may be difficult to represent and manipulate on the computer. We therefore construct a discrete outer representation $R$ using a uniform grid of subboxes on $[0,1]^2$, which like Ceberio et al. [22] we store with a $n \times n$ bit matrix. $\mathcal{R}$ may also be extended to allow for interval inputs, in which case $\mathcal{R}$ returns a 1 if *any* real values in the interval are included in $\mathcal{R}$

$$R(u, v) = \begin{cases} 1 & \text{if } (u \times v) \cap \mathcal{R} \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Stated another way, $R$ returns a 0 only if all the values in the interval are not in the original set, and in that sense $R$ guarantees *exclusion*. The function $R$ can be represented by a $n^2$ bit-matrix by subintervalising $[0,1]^2$ and evaluating Eq. (3) on each sub box, forming a discrete outer approximation of the original dependence $\mathcal{R}$. Note the functions $\mathcal{R}$ and $R$ are strongly related to *binary relations* from set theory, which associate elements of one set to elements of another. For this reason we label functions like $R$ as relations or interval relations, as they relate a particular subinterval in one dimension to a collection of subintervals of another dimension. Fig. 3 shows a hypothetical interval relation represented by a bit-matrix, with some examples of intervals which would be included and excluded. The set is the union of two rings, and is just an example of a generic bivariate set picked to show the flexibility of this discrete set representation. Note that the relation does not need to extend to the entire $[0,1]^2$ range, which is a modelling choice. Information about the bounds of one variable may inform the bounds of the other, and may not extend to its entire range. Also for practical reasons, any values outside $[0,1]^2$ will be returned as 0.

Similar to how a bivariate distribution may be defined in terms of two marginals and a copula, a joint interval may be defined by two marginal (or univariate) intervals $X_1, X_2$ and a relation $R(u, v)$, by $J_{X_1,X_2}(x, y) = \text{supp}\, R(u, v)$ with $u = (x - \underline{X_1})/(\overline{X_1} - \underline{X_1})$ and $v = (y - \underline{X_2})/(\overline{X_2} - \underline{X_2})$. This defines a grid of subboxes following $R$ by a uniform scaling by the intervals $X_1$ and $X_2$, and then taking the support of resulting indicator function. This allows dependencies to be stored and manipulated independently from marginal intervals. Similar to how univariate intervals bound a sets of probability distributions, i.e. an interval is a special case of a p-box, these joint intervals may be considered to bound a set of joint distributions: the set of bivariate distributions whose ranges are a subset of the bivariate set.

Some useful special cases of interval relations are shown on Fig. 4. Shown is non-interactivity or Fréchet, which is the standard interval case which make no dependence assumptions. In perfect, opposite and functional associations one variable is completely determined by the other, and are useful in arithmetic since intervals resulting from simple scalar transformations and scaling will produce perfect or opposite dependence. Univariate transformations will give functional dependencies. This, and arithmetic with generic binary and unary operations, is discussed in the next section.

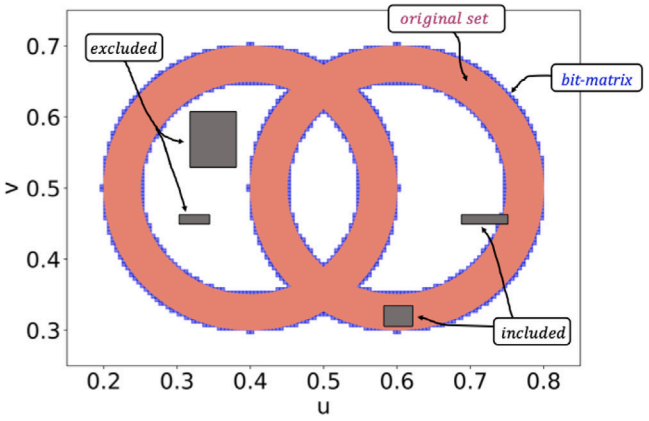


**Fig. 3.** Shows an outer approximation of a relation represented by a bit-matrix. Also shown are several intervals which would be included and excluded by the relation.

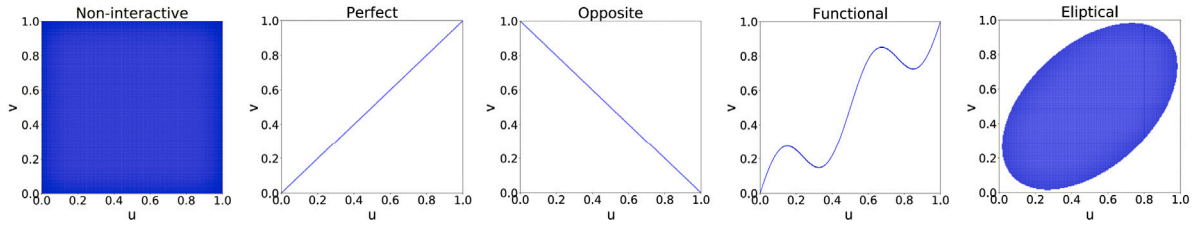


**Fig. 4.** Some special cases of interval relations.

## 3. Relation arithmetic

Ceberio et al. [22] suggest that binary operations are performed with $J$ using interval arithmetic on each individual subbox, which requires of the order $\sim n^2$ interval operations. We proposed an alternative method which requires $\sim n$ operations. For this we borrow another concept for probability theory: *conditioning*. Much like how a univariate distribution can be created from a bivariate distribution by conditioning on a value, we produce a conditional (univariate) set from a bivariate set by constraining it to particular values. A conditional set $J(x|\ y = Y)$ may be created from the bivariate interval $J(x, y)$ by intersecting the it with an interval, or real value, $Y$

$$J(x|\ y = Y) = J(x, y) \cap ([-\infty, \infty] \times Y). \tag{4}$$

$J(x|\ y = Y)$ is the conditional sets indicator function, which may be the union of several disjoint intervals (an interval union [24]). For example, if we condition the relation in Fig. 3 with $v = [0.48, 52]$, we would get a set made up of 4 disjoint intervals: $[0.195, 0.255] \cup [0.395, 0.455] \cup [0.545, 0.605] \cup [0.745, 0.8]$. Fig. 5 shows an illustration of this. Calculations of this sort can be performed quite efficiently with a bit-matrix representation, since it is a simple subselection of arrays of $R$.

We perform dependent interval operations using this conditioning. Instead of $n^2$ interval operations, we perform $n$ (for each row or column) set operations on the conditionals. Since a conditional may contain more than one interval, multiple interval operations are performed per set operation, however the overall number of interval operations remains lower than $n^2$. The minimum number of interval operations is $n$, corresponding to the non-interactive case (everywhere *true*). The maximum number of required interval operations is $n^2/2$, which is the very extreme case where every other element of the bit matrix is empty.

Using this conditional arithmetic, the relations between inputs and outputs of operations may be computed. Take for example the simple sequence of operations shown earlier: $x + y = z$, and then $z - x$, where initially the joint interval $J_{XY}$ is known. For $z - x$ to be evaluated tightly, the joint $J_{ZX}$, or similarly the relation $R_{ZX}$, must be calculated. For this we condition $J_{XY}$ on some subinterval $x = X$, and evaluate the $x + y$ with the conditional set, giving $J_{ZX}(z|x = X)$. This is repeated for $n$ subintervals of $x$, and the joint $J_{ZX}$ is constructed as the union of all conditionals of $z$

$$J_{ZX}(z|x = X) = J_{YX}(y|x = X) + X, \tag{5}$$

$$J_{ZX}(z, x) = \bigcup_{i=1}^{n} J_{ZX}(z|x = X_i), \tag{6}$$

where the $+$ operator is an interval sum, and where $X_i$ is a subinterval. Fig. 6 shows an example of this, where $x + y = z$ is evaluated for $x, y = [-1, 1]$ and with $J_{XY}$ having the complement of a ring shaped dependence shown on the left. The calculated

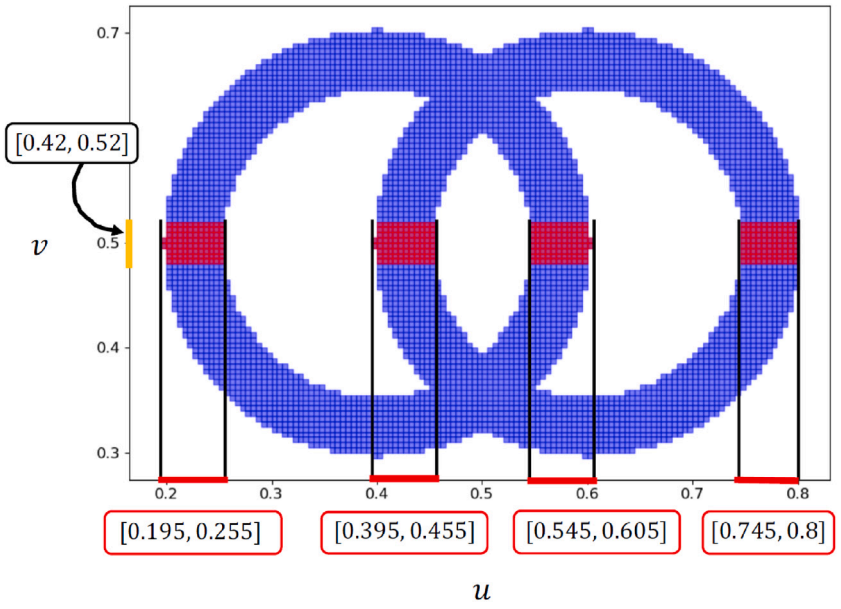


**Fig. 5.** Illustration of conditioning a relation on the interval $v = [0.48, 52]$. The resulting set is the union of the disjoint intervals $[0.195, 0.255] \cup [0.395, 0.455] \cup [0.545, 0.605] \cup [0.745, 0.8]$.

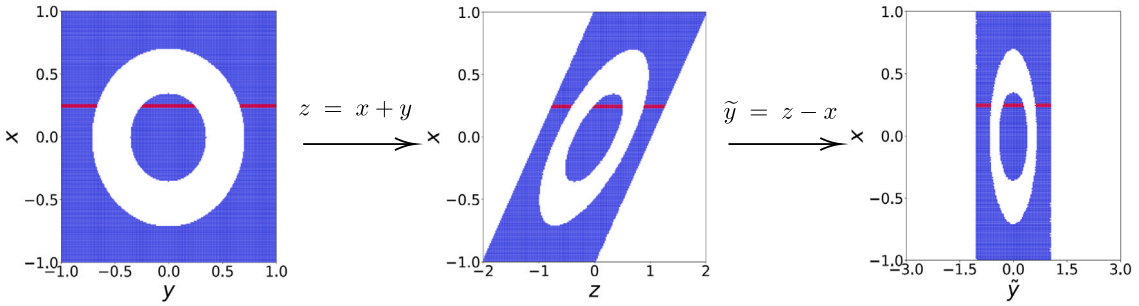


**Fig. 6.** An example of using conditional arithmetic to propagate an interval relation though a binary operator. Shown is $x + y = z$, with $x, y = [-1, 1]$ having the dependency shown on the left figure. Centre shows the calculated bivariate set $J_{ZX}$. When the inverse operation of $\widetilde{y} = z - x$ is performed, the original $J_{XY}$ is retrieved. Shown in red is a conditional set, conditioned on a particular subinterval of $X$.

interval $z = [-2, 2]$ as expected, however the dependence has also been propagated. If $\widetilde{y} = z - x$ is evaluated, then $\widetilde{y} = [-1.08, 1.08]$, which tightly encloses the exact solution of $\widetilde{y} = y = [-1, 1]$, and a vast improvement of the standard interval arithmetic answer of $\widetilde{y} = [-3, 3]$. Moreover, when the inverse operation is performed, and outer approximation of the initial dependence is retrieved $J_{XY} \subseteq J_{X\widetilde{Y}}$ The tightness of the result depends on the discretisation used, and for this example a discretisation $200 \times 200$ was used. Using fewer subboxes will compute the result quicker, but giving a wider result. The method however is still guaranteed to bound the exact relation and marginals. An identical calculation can be performed for $J_{ZY}$, conditioning on subintervals of $Y$. The above shows sum as an example, any binary operation $\circ$ that has an interval extension (e.g. $\circ \in \{+, -, \times, \div, \min, \max\}$) can be used:

$$J_{X\circ Y,X}(z, x) = \bigcup_{i=1}^{n} J_{YX}(y|x = X_i)\circ X_i \,, \tag{7}$$

$$J_{X\circ Y,Y}(z, y) = \bigcup_{i=1}^{n} J_{XY}(x|y = Y_i)\circ Y_i \,. \tag{8}$$

### 3.1. Unary operations

Dependency can be tracked in a similar way through a unary operation with an interval extension $f : \mathcal{B} \to \mathcal{B}$. To determine the joint interval between an input and its transformation $f(X)$, we again condition $X$ on various subintervals, and take the union of

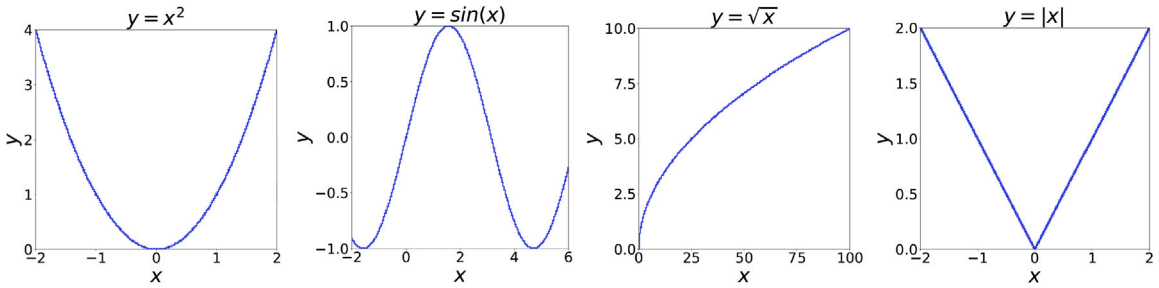


**Fig. 7.** Examples of dependency tracking through unary operations.

the images:

$$J_{f(X),X}(z,x) = \bigcup_{i=1}^{n} f(J_X(x = X_i)) \,. \tag{9}$$

This is identical to the calculation that is performed for standard subintervalisation, where $X$ is split into linearly spaced subintervals $X_i$, and $f$ is evaluated for each subinterval. The bit-matrix representing the relation $R_{f(X),X}$ is then filled with 1s where there is an overlap between the calculated boxes $f(X_i) \times X_i$ and a uniform grid of boxes in the domain $f(X) \times X$. Fig. 7 shows examples of bivariate intervals constructed using various unary transformations.

### 3.2. Chained operations

The above arithmetic gives the dependencies for intervals linked by binary operations $J_{X \circ Y,X}$, or unary transformations $J_{f(X),X}$. We can therefore use these dependencies to further propagate intervals in expressions composed of chained operations, such as $W = X\sin(X)$ or $W = (X+Y)X$. The propagation problem for $X\sin(X)$ can be solved by first performing $\widetilde{W} = \sin(X)$, and calculating $J_{\sin(X),X}$, followed by a multiplication $W = \widetilde{W} * X$ using $J_{\sin(X),X}$. The dependency between $J_{W,X}$ may also calculated.

Similarly $W = (X+Y)X$ may be chained as $\widetilde{W} = X+Y$ (finding $J_{X+Y,X}$), followed by a binary $W = \widetilde{W} * X$. An initial dependency for $J_{XY}$ may be specified and propagated. Fig. 8 shows an example of propagating an elliptical relation through repeated evaluations of $w_{i+1} = w_i * x$. It can be seen that the set does not remain an ellipse. A similar calculation is shown in Fig. 9 but with an initial ring shaped dependence with two different resolutions. The top row in shows a fine resolution of $n = 200$, and a coarser $n = 20$ shown on the bottom row. The lower fidelity calculation is guaranteed to enclose the higher fidelity one, and any calculation which is performed with an $n > 20$. The whitespace of these calculations is guaranteed to be excluded from the propagated bivariate set at any resolution.

## 4. Interval relation networks

The previous section describes a method to propagate dependencies in variables linked by operations. However, generally not all variables in a program will be linked by an operation. Even for simple expressions such as $K = (X + Y)Z$, the first operation $W = X + Y$ may be performed exactly, giving $J_{WX}$ and $J_{WY}$, however for the following $K = W * Z$, $J_{WZ}$ is required. Certainly $W * Z$ may be performed with an unknown dependence (the standard interval case), but it is possible to do better. Given that $J_{WX}$ (or $J_{WY}$) and $J_{XZ}$ (or $J_{YZ}$) is known, what can be said about $J_{WZ}$? We show that a surprising amount of information about $J_{WZ}$ can be found from known dependencies such as $J_{WX}$ and $J_{XZ}$.

The goal of this work is also to produce a method which is automatic. At the beginning of one's program, uncertain variables should be specified as intervals with any proceeding code tightly evaluated with arithmetic, with little or no consideration from the user about the underlying complexities of the method. We therefore require a storage model for the known dependencies at a particular point in a programs execution, such that when a binary operation is called, the correct dependence may be retrieved and used. As operations are performed, and new dependencies calculated, they should also be stored appropriately for future use. If a particular dependence is required but is not available, it must be automatically inferred from the known dependencies.

For this task, we borrow a final model from probability: a *vine copula* [25]. A vine copula is a method for constructing higher dimensional copulas solely from pairwise bivariate marginal information. An undirected network is constructed, with nodes as random variables and edges as pairwise copulas. Under certain assumptions, such as conditional independence and with only specific network topologies being allowed, a high dimensional dependence structure can be constructed. We construct a similar network model, but with intervals as nodes and bivariate relations as edges. The proposed interval relation networks have no constraints on the topology and requires no additional assumptions.

As an illustrative example, consider the expression $K = \sin(XY)(XZ)$. Say that all three input intervals are known $X = [1, 3]$, $Y = [-1, 2]$, and $Z = [-2, 1]$, as well as the pairwise dependencies $J_{XY}$ and $J_{YZ}$. In a program, this expression would be evaluated by the following sequence of operations:

$$W_1 = X * Y \qquad \text{(requires } J_{XY}\text{, gives } J_{W_1X} \& J_{W_1Y}\text{)}$$

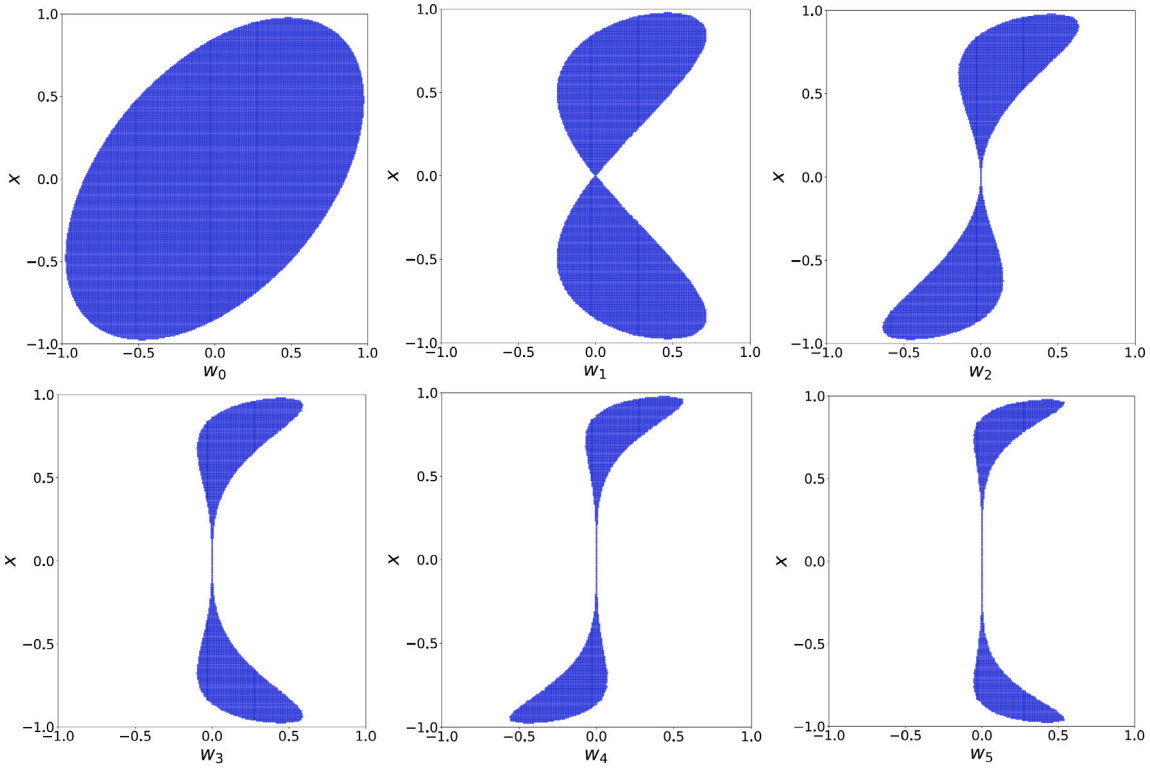


**Fig. 8.** Shown is the evolution of an ellipse through the repeated evaluation of $w_{i+1} = w_i * x$.

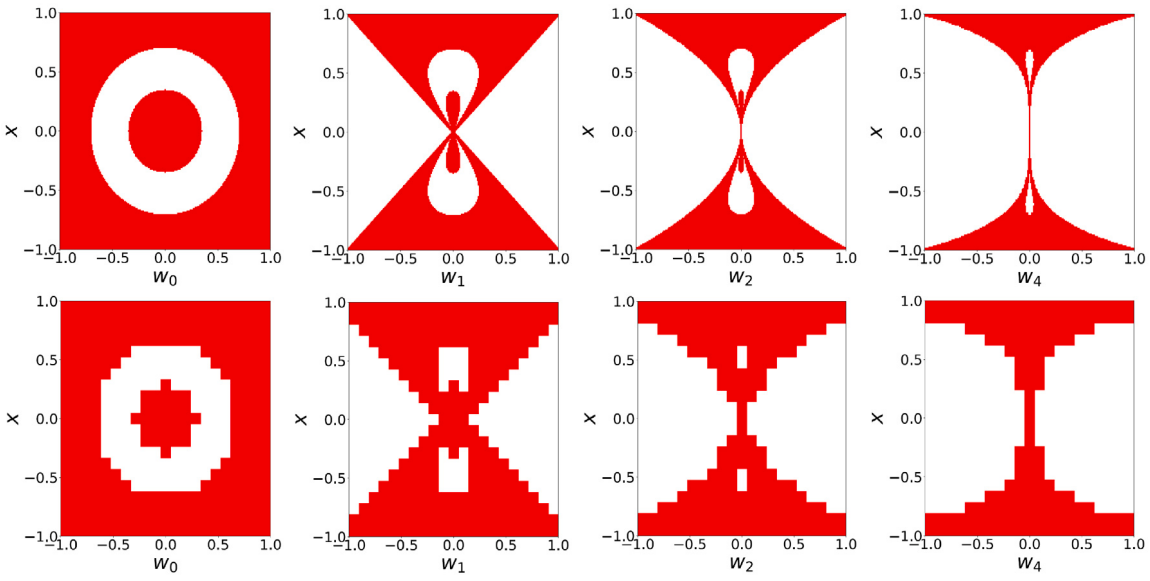


**Fig. 9.** Shown is the evolution of an interval relation with two different resolutions through the repeated evaluation of $w_{i+1} = w_i * x$, starting from the complement of a ring shaped relation in the left. The top row shows a fine resolution of $n = 200$, and the lower row shows a coarse $n = 20$.

$$W_2 = \sin(W_1) \quad \text{(gives } J_{W_2W_1}\text{)}$$
$$W_3 = X * Z \quad \text{(requires } J_{XZ}\text{, gives } J_{W_3X} \& J_{W_3Z}\text{)}$$
$$K = W_3 * W_2 \quad \text{(requires } J_{W_3W_2}\text{).}$$

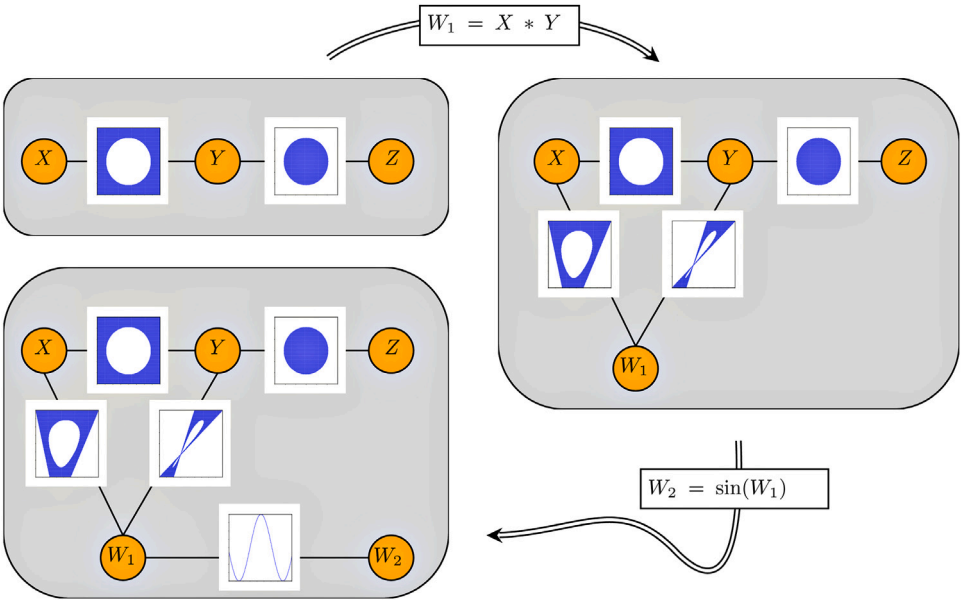


**Fig. 10.** Shows the evolution of an interval relation network through a sequence of calculations. The network stores interval variables (nodes) and known relations (edges) at a particular point in the program. New intervals and relations are added as the calculation executes.

The initial state of the program, where $X$, $Y$, $Z$, $J_{XY}$, and $J_{YZ}$ are known, is shown on the top left of Fig. 10. The nodes of this network store the intervals $X = [1, 3]$, $Y = [-1, 2]$, and $Z = [-2, 1]$, and the links store the known two dependencies $J_{XY}$ and $J_{YZ}$. For the sake of this example, we begin with $J_{XY}$ being a complement of a circle and $J_{YZ}$ being a circle. Fig. 10 shows the evolution of the interval relation network corresponding to the first two steps the evaluation of the expression. The first step $W_1 = X * Y$, requiring $J_{XY}$ which is known by the network and can be evaluated, results in $W_1 = [-3, 6]$. This is the same result as standard interval arithmetic, however additionally $J_{W_1X}$ and $J_{W_1Y}$ are calculated. After this binary operation, $W_1$ is added as a new node and is linked to $X$ and $Y$ using $J_{W_1X}$ and $J_{W_1Y}$. The network's state after this operation is shown on the top right of Fig. 10, showing the newly calculated relations.

The next unary operation $W_2 = \sin(W_1)$ only requires the marginal $W_1$ known from the previous step. The calculated $W_2 = [-1, 1]$ is added and linked to $W_1$ using $J_{W_2W_1}$ which encodes the evaluated sin function. The proceeding $W_3 = X * Z$ requires $J_{XZ}$, which is not currently available in the network. However $J_{XY}$ and $J_{YZ}$ are known, and may be used to bound $J_{XZ}$. How such inferences can be performed is discussed in the next section.

### 4.1. Relation inference

Much like how in a vine copula, where the two dimensional marginals are defined and used to construct a higher dimensional copula, the two dimensional sets of the interval network are the two dimensional marginals (projections or shadows) of a higher dimensional set. Indeed the network shown at the bottom left in Fig. 10 with five intervals has a five dimensional set associated with it, but all that is known about this shape are five of its shadows $J_{XY}$, $J_{YZ}$, $J_{W_1X}$, $J_{W_1Y}$, and $J_{W_2W_1}$. In most cases there will not exists a unique higher dimensional set which has these 2D shadows. However, it is possible to construct a outer bound on all 5D shapes with these marginals, and use this higher dimensional shape to determine the required dependencies by projecting it onto the axis planes. In the case where there is no high dimensional shape with these projections exits, this should also be determinable and returned to the user as an error. However this will only occur at the beginning of the calculation with the user defined dependencies, and will not occur as a result of a sequence of operations.

Unlike vine copulas, there are no additional assumptions required to construct these higher dimensional sets. For copulas, the problem of finding an n-copula from two dimensional marginals is well studied. Only in very limited situations is it possible to find a unique n-copula from the marginals, with solutions often being vacuous (no better than the Fréchet bounds). In vines, certain assumptions are made to return a single n-copula with the desired margins, but it will not necessarily be the only solution. In the presented interval relation networks, no such assumptions are needed. The returned result will be a rigorous bound on the n-d set and projections, and in most cases will not be vacuous. Some higher dimensional information is lost by only considering the projections, for example a cube containing an empty sphere cannot be represented this way. But crucially the presented framework is sufficient to determine the required dependencies well enough to reduce repeated variable problems.

Constructing such higher dimensional sets every time a dependence inference is required would be very computationally complex. Therefore we only ever perform this calculation in three dimensions, i.e. in the determination of $J_{XZ}$ from $J_{XY}$ and $J_{YZ}$, we only

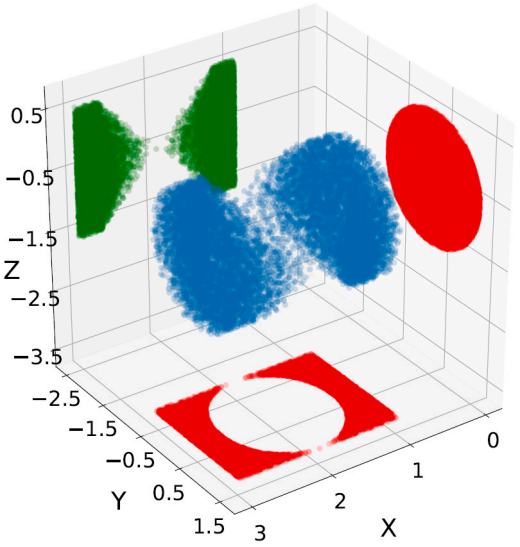


**Fig. 11.** Visualisation of dependence inference using two known bivariate projections (red) to determine a third (green).

project in three dimensions $J_{XYZ}$. It is easy to visualise this projection inference process with sampling, shown in Fig. 11. The volume defined by the marginals $X \times Y \times Z$ has been uniformly sampled, and the samples which fall outside the sets $J_{XY}$ and $J_{YZ}$ are rejected, giving the blue 3D scatter in the centre and the red $XY$ and $ZY$ projections. The accepted samples can then be projected onto the required $XZ$ axis giving the missing dependence, shown in green. Certainly, using sampling is not rigorous and will lead to some missing features of the inferred dependence (in low volume regions), but Fig. 11 serves as a useful visualisation of this process.

Interval relation inference can be performed rigorously and quite efficiently in terms of bit-matrices. Like for relation arithmetic, this step can be performed with conditioning. The sets $J_{XY}$ and $J_{YZ}$ have the common variable $Y$, which will be conditioned on in both sets. Conditioning both sets on the same subinterval of $Y_i$ gives two univariate sets $J_{XY}(x|y = Y_i)$ in $X$, and $J_{YZ}(z|y = Y_i)$ in $Z$. The Cartesian product of these sets $J_{XY}(x|y = Y_i) \times J_{YZ}(z|y = Y_i)$ exists in the $XZ$ plane, and defines a part of $J_{XZ}$. Taking the union for all subintervals of $Y$, the dependence $J_{XZ}$ is found

$$J_{XZ} = J_{XY} \circledast J_{YZ} = \bigcup_{i=1}^{n} J_{XY}(x|y = Y_i) \times J_{YZ}(z|y = Y_i)\,. \tag{10}$$

In a sense, the common variable $Y$ is marginalised out. The operator $\circledast$ denotes this set based inference using relations with a common variable. The operator $\circledast$ cancels out the common variable, the right variable in the first operand, and the left variable in the second, i.e. $J_{X_1X_2} \circledast J_{X_2X_3}$ marginalises out $X_2$ and combines $X_1, X_3$ to give $J_{X_1X_3}$. The right of Fig. 12 shows the result of $J_{XY} \circledast J_{YZ}$, and is the same as the green projection that was found by sampling in Fig. 11. It is simple to check that all trails by sampling fall in the set produced by $J_{XY} \circledast J_{YZ}$. In red are the sets found from conditioning on a particular subinterval of $Y$, which is used as a part of the construction of $J_{XZ}$. This process may quite efficiently be performed with bit-matrices, where the same array of $R_{XY}$ and $R_{YZ}$ is subselected. The Cartesian product of the indices of those arrays which have value 1 are then filled in $R_{XZ}$. A Julia function for performing this projection step is available in Appendix.

This relation inference allows us to continue the calculation of $K = \sin(XY)(XZ)$. The continuation of the associated network is shown in Fig. 13, with the network containing newly calculated $J_{XZ}$ shown on the top right. The following $W_3 = X * Z$ may then be evaluated, with the new variable and relations added to the network, shown on the bottom left of Fig. 13. The final operation in the calculation is $W_3 * W_2$, requiring $J_{W_3W_2}$. The variables $W_3$ and $W_2$ do not share a common neighbour in the network, and so the inference cannot be performed directly. However there are paths in the network which link the two nodes, and so it is possible to perform several chained inference steps to determine $J_{W_3W_2}$.

### 4.2. Chained inference

The inference operator $\circledast$ may be chained through several relations which have common variables, i.e. any path in the interval network which links two nodes. For example, one of the paths linking $W_2$ and $W_3$ is $W_2 \to W_1 \to Y \to Z \to W_3$, which gives the following chained inference

$$J_{W_2W_3} = J_{W_2W_1} \circledast J_{W_1Y} \circledast J_{YZ} \circledast J_{ZW_3}. \tag{11}$$

There is not a unique way to solve the above expression. For example, always solving from the left gives

$$J_{W_2W_3} = (J_{W_2W_1} \circledast J_{W_1Y}) \circledast J_{YZ} \circledast J_{ZW_3}$$

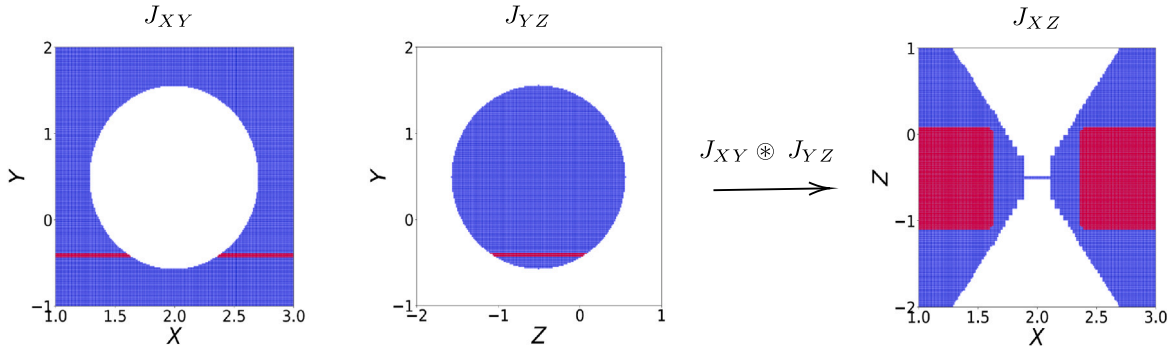


**Fig. 12.** Shows the interval relation inference of $J_{XZ}$ from $J_{XY}$ and $J_{YZ}$. In red is a particular conditional set, conditioned on a subinterval of $Y$, which is used to determine part of $J_{XZ}$.

$$
\begin{aligned}
&= (J_{W_2Y} \circledast J_{YZ}) \circledast J_{ZW_3} \\
&= (J_{W_2Z} \circledast J_{ZW_3}) = J_{W_2W_3},
\end{aligned}
$$

which gives the new dependencies $J_{W_2Y}$, $J_{W_2Z}$, and $J_{W_2W_3}$. This leads to the final network configuration shown on the bottom right of Fig. 13. This is however not the only way Expression (11) can be solved. Solving always from the right

$$
\begin{aligned}
J_{W_2W_3} &= J_{W_2W_1} \circledast J_{W_1Y} \circledast (J_{YZ} \circledast J_{ZW_3}) \\
&= J_{W_2W_1} \circledast (J_{W_1Y} \circledast J_{YW_3}) \\
&= (J_{W_2W_1} \circledast J_{W_1W_3}),
\end{aligned}
$$

which still gives $J_{W_2W_3}$, but also gives the different dependencies $J_{YW_3}$ and $J_{W_1W_3}$. Indeed one could solve from the centre, or any combination of these. It is difficult to determine *a priori* which one of the ways will be most efficient (gives the tightest $J_{W_2W_3}$), so as a heuristic we always solve from the left. Irregardless of the path taken, the calculation will give a rigorous bound on the dependence. Some discussion of making use of these various calculation paths is discussed in Section 6. However, an interesting feature of this is that this chained inference step can be parallelised. Consider the following chained calculation, with this time the various brackets solved concurrently

$$
\begin{aligned}
J_{X_1X_9} &= (J_{X_1X_2} \circledast J_{X_2X_3}) \circledast (J_{X_3X_4} \circledast J_{X_4X_5}) \circledast (J_{X_5X_6} \circledast J_{X_6X_7}) \circledast (J_{X_7X_8} \circledast J_{X_8X_9}) \\
&= (J_{X_1X_3} \circledast J_{X_3X_5}) \circledast (J_{X_5X_7} \circledast J_{X_7X_9}) \\
&= (J_{X_1X_5} \circledast J_{X_5X_9}).
\end{aligned}
$$

The serial calculation would have taken seven steps, where the parallel one only takes three.

The path $W_2 \to W_1 \to Y \to Z \to W_3$ is also not the only path linking $W_1$ and $W_3$. Looking at the network in the bottom left of Fig. 13, one can see that the following are also possible paths:

$$
\begin{aligned}
&W_2 \to W_1 \to X \to Y \to Z \to W_3, \\
&W_2 \to W_1 \to Y \to X \to W_3, \\
&W_2 \to W_1 \to X \to W_3.
\end{aligned}
$$

Indeed, the last of these appears to be the shortest, however the path $W_2 \to W_1 \to Y \to Z \to W_3$ is actually preferred since it gives the tightest inference. We therefore require a method to automatically find and select the best path for chained inference. In principle any standard network path finding algorithm could be used, for example Dijkstra's algorithm or the $A^*$ algorithm. In this work the $A^*$ algorithm was used, but with no particular preference over Dijkstra's algorithm. We do however add weights or distances between nodes in the graph search as a heuristic to get tighter inferences. Consider for example the network shown at the top of Fig. 14, with five variables $X_1, X_2, X_3, X_4, X_5$, and with $J_{X_1X_2}, J_{X_2X_3}, J_{X_3X_5}$ as perfect dependencies, and with $J_{X_1X_4}$ and $J_{X_4X_5}$ as non-interactive. Fig. 14 shows two inferences of $J_{X_1X_5}$, the centre one using the path $X_1 \to X_2 \to X_3 \to X_5$, and the bottom using a shorter $X_1 \to X_4 \to X_5$. Although it is a longer path, the first inference gives a much more precise dependence, perfect dependence as opposed to non-interaction from the short path. This is clearly preferred since it gives tighter arithmetic. We can encode this preference by giving weights or distances to the links in the network, which the path finding algorithm will use in its determination of the shortest path. A simple distance metric is the *area* in the set compared to the bounding box's area, i.e. blue area divided by the total area. In terms of a bit-matrix representation of $R_{X_1X_2}$, this is a simple sum of all elements with value 1 over the total number of elements in the bit-matrix

$$
\text{dist}(X_1, X_2) = \frac{\sum_{i,j}^{n} R_{X_1X_2}[i,j]}{n^2}. \tag{12}
$$

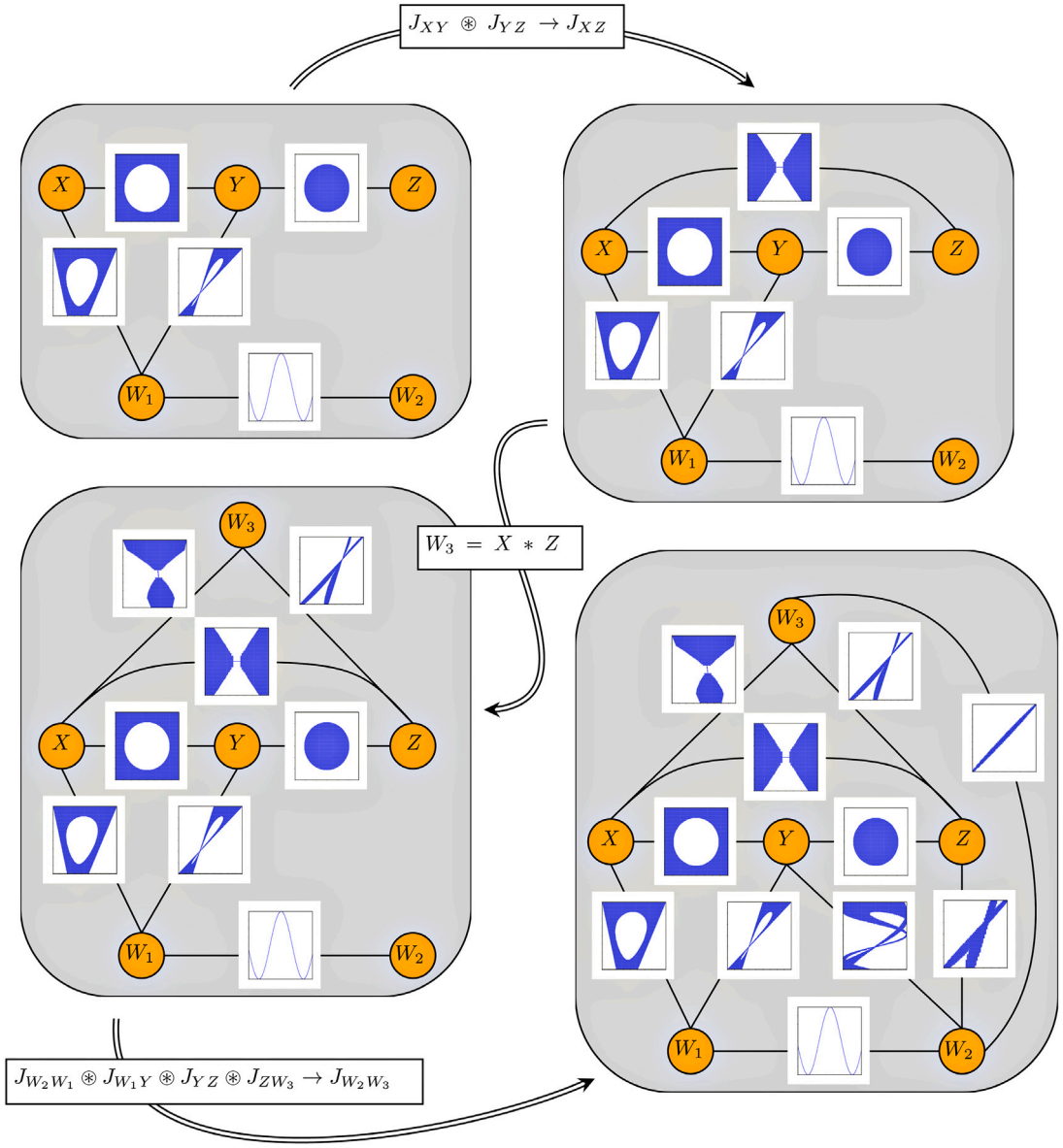


**Fig. 13.** Shows the evolution of an interval relation network including several relation inference steps. The first step infers $J_{XZ}$, from $J_{XY}$ and $J_{YZ}$, which is added to the network as a link. The next steps is a multiplication involving the new $J_{XZ}$, which adds a new node $W_3$ and two newly calculated relations. The final steps shows a chained inference of $J_{W_2W_3}$ through a path linking the nodes, which adds several new relations.

Very precise dependencies (such as perfect or opposite) will give distances close to 0, and non-interactivity will be assigned the maximum distance of 1. This gives the path $X_1 \rightarrow X_4 \rightarrow X_5$ a total distance of $\text{dist}(X_1, X_4) + \text{dist}(X_4, X_5) = 1 + 1 = 2$, and gives $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_5$ a total distance of $0.02 + 0.02 + 0.02 = 0.06$ (for an $n = 50$). When these distances are incorporated, the path finding algorithm correctly identifies $J_{X_1X_2} \circledast J_{X_2X_3} \circledast J_{X_3X_5}$ as the best inference.

A question can arise about which path is optimum in a network consisting solely of perfect and opposite dependencies. In a network configuration comprised entirely of perfect dependencies, the algorithm would identify the shortest (in terms of number of nodes) as the tightest path. In fact, in this situation it does not matter which path is chosen, any inferred dependencies would correctly be identified as perfect. The situation is slightly more complex in a configuration of a mix of perfect and opposite. The algorithm cannot distinguish paths of equal area, but often these configurations are not mathematically possible. Considering a network with three variables $X$, $Y$, and $Z$, a configuration of $J_{XY}$ = perfect, $J_{YZ}$ = opposite, and $J_{XZ}$ = opposite is not valid [26]. For correlations, this gives a correlation matrix which is not semi-positive definite. In copulas, there is no higher dimensional copula with these 2-copula marginals. For relations, there is no 3D set that has these exact projections. This is a well-studied problem in

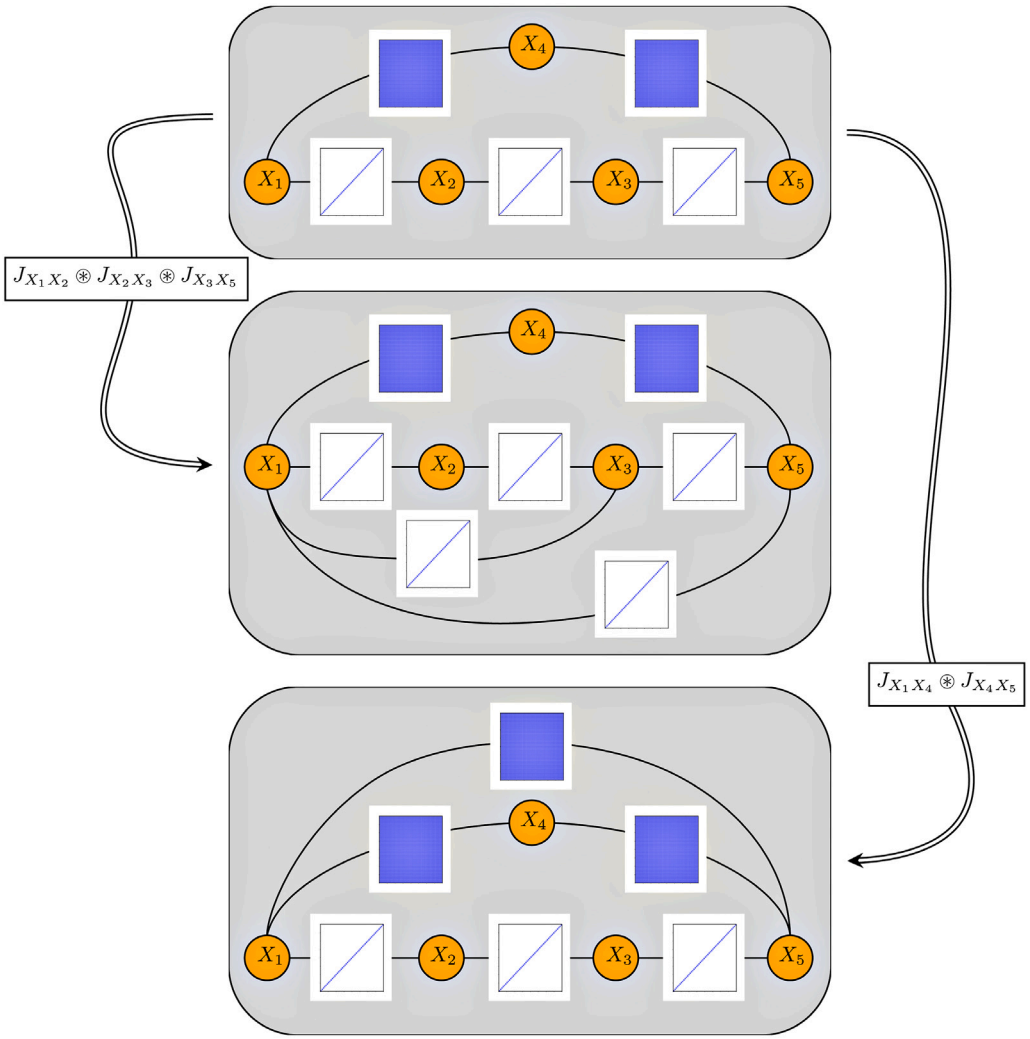


**Fig. 14.** Shows two different inferences of $J_{X_1X_5}$. The network in the centre uses $J_{X_1X_2} \circledast J_{X_2X_3} \circledast J_{X_3X_5}$, giving a perfect dependence between $J_{X_1X_5}$. The bottom shows $J_{X_1X_4} \circledast J_{X_4X_5}$ giving non-interaction. Although the second inference is from a shorter path, the first gives a tighter relation and is preferred.

copulas called compatibility, and these sorts of non-compatibility bivariate dependencies cannot arise as a result of elementary arithmetic operations.

Returning to the calculation of $K = \sin(XY)(XZ)$, the resulting $J_{W_2W_3}$ is surprisingly precise, a near straight line resembling perfect dependence. Evaluating the final $K = W_2 * W_3$ gives a $K = [-0.24, 6]$ (using a resolution of $n = 200$ throughout) and a clear improvement over standard arithmetic of $[-6, 6]$. Importantly the method can also obtain any dependencies between variables, for example the output and the inputs, if further propagation is required. The calculation was serial and took 0.9s, including the initial dependency assignment, network creation and inference, on a mid-range 2019 laptop computer (2.3 GHz Quad-Core Intel Core i5 and 8 GB RAM), implemented in the Julia programming language.

## 5. Applications

The method presented in this paper has been implemented in a open source Julia package `RelationArithmetic.jl`[1]. The software uses `IntervalArithmetic.jl` [27] for the base arithmetic, and `Graphs.jl` [28] and `MetaGraphs.jl` [29] for the

[1] https://github.com/AnderGray/RelationArithmetic.jl

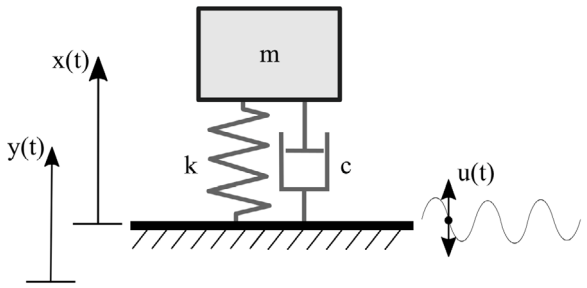


**Fig. 15.** Mass-spring-damper system with base acceleration.

network model. All of the presented applications have been performed by writing a deterministic function in Julia, specifying some intervals at the beginning of a script, and using `RelationArithmetic.jl` to automatically track dependencies. As an example, the following script produces the final plot of the elliptical relation example in Fig. 8.

```
using RelationArithmetic

x = intervalR(-1, 1)  # R for relation
w = intervalR(-1, 1)

R = ellipse(e = 0.5, n = 200)  # eccentricity and resolution
define_relation(x, w, R)  # gives intervals specified dependence

function iterate5(w, x) # define a function
  for i = 1:5
   w = w * x
  end
  return w
end

w_5 = iterate5(w, x)  # pass intervals to function
plot(w_5, x)  # plot bivariate set
```

### 5.1. Forced harmonic oscillator

This example performs interval arithmetic in a dynamical system shown in Fig. 15, a damped oscillator subject to harmonic ground motion. The quantity of interest is the displacement amplitude $|y|$, with the mass $M$, stiffness $K$, and damping coefficient $C$ as intervals. The absolute displacement is

$$y(t) = x(t) + u(t). \tag{13}$$

It is assumed that (i) the inertial forces are proportional to the acceleration of the mass $m$; (ii) the damping force and the elastic force are proportional to the relative velocity $\dot{x}$ and the relative displacement $x$, respectively. The equation of motion is

$$M\ddot{y} + C\dot{x} + Kx = 0. \tag{14}$$

The equation of motion in terms of absolute displacement is

$$M\ddot{y} + C\dot{y} + Ky = Ku + C\dot{u}. \tag{15}$$

If $u(t)$ is harmonic, the solution to Eq. (15) can be obtained substituting the two exponentials

$$u(t) = u_0\, e^{j\omega t}, \qquad y(t) = y\, e^{j\omega t},$$

which when placed into Eq. (15) give

$$(-M\omega^2 + j\omega C + K)\, y\, e^{j\omega t} = (K + j\omega C)\, u_0\, e^{j\omega t},$$

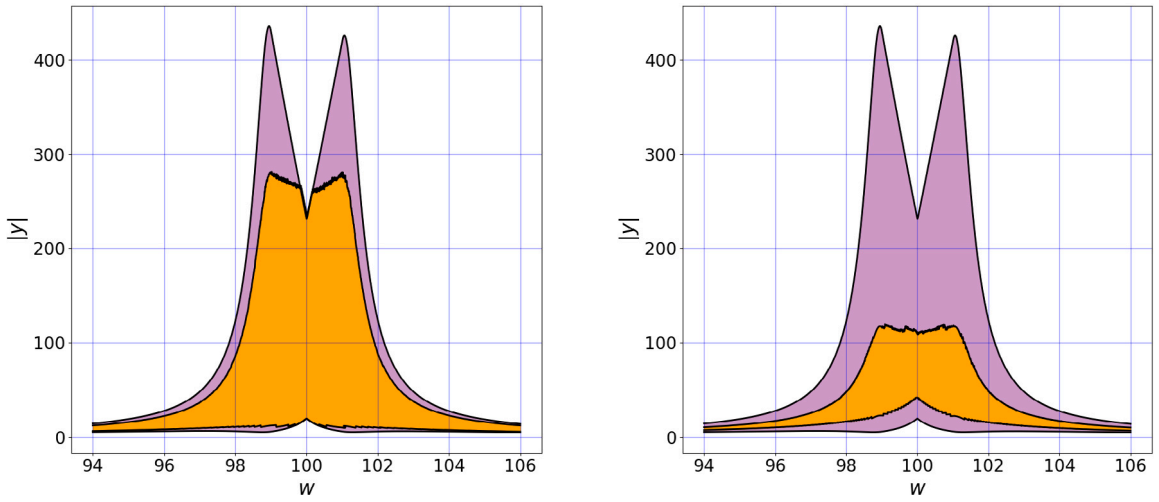


**Fig. 16.** Amplitude spectrum bounds of an oscillator subject to harmonic ground motion, with frequency $w$ and with interval parameters. In purple shows `IntervalArithmetic.jl`, and `RelationArithmetic.jl` is in orange. The right figure shows the bounds with the assist of variable reuse.

from which we obtain the displacement as

$$y = u_0 \frac{K + j\omega C}{-M\omega^2 + j\omega C + K} . \tag{16}$$

The displacement of Eq. (16) is complex (harmonic wave), so it carries information about the amplitude and phase. The absolute value $|y|$ is the amplitude of the harmonic wave, which will be the quantity of interest in this example. Setting the amplitude of the ground force motion to one $u_0 = 1$, and taking the absolute value of (16) gives

$$|y| = \frac{\sqrt{\left(-\omega^2\ M\ K + K^2 + (\omega\ C)^2\right)^2 + \left(-\omega^3\ C\ M\right)^2}}{\left(-\omega^2\ M + K\right)^2 + (\omega\ C)^2} . \tag{17}$$

We may now perform interval arithmetic through the above expression. For the intervals

$$\begin{aligned} M &= [9900, 10100] \quad \text{(kg)} , \\ K &= [99000000, 10100000] \quad \text{(N/m)} , \text{ and} \\ C &= [9900, 10100] \quad \text{(N s/m)} \end{aligned}$$

we obtain the results in left of Fig. 16. Purple shows the bounds obtained with `IntervalArithmetic.jl`, and orange shows the results using `RelationArithmetic.jl`. The improvement in the bounds is significant, the interval widths are about halved. For this calculation Expression (17) was essentially copied into Julia, intervals specified and propagated using `RelationArithmetic.jl`. We can however do better by reusing certain variables in the implementation. For example, setting $A = -\omega^2\ M + K$, and realising that $-\omega^2\ M\ K + K^2 = A * K$. This gives the following expression:

$$|y| = \frac{\sqrt{\left(A * K + (\omega\ C)^2\right)^2 + \left(-\omega^3\ C\ M\right)^2}}{A^2 + (\omega\ C)^2} , \tag{18}$$

with $A = -\omega^2\ M + K$. The result of evaluating the expression like this is seen on the right of Fig. 16, with the bounds from `RelationArithmetic.jl` being much tighter. The reason reusing variables is tighter is we have directly specified that the *same* $A$ appears twice, and helps the method identify dependencies. When Expression (17) was evaluated, two instances of $A$ were made, $A_1 = -\omega^2\ M + K$ and $A_2 = -\omega^2\ M + K$, two different nodes in the dependence network. These two variables are actually the same interval (have the same bounds and are perfectly dependent), however this can be difficult to determine. The method instead constructs an outer approximation to this precise perfect dependence, which is why the left of Fig. 16 is wider. With Expression (18) this is implicitly stated, and allows for the perfect dependence to be precisely specified, resulting in tighter intervals. In the current implementation of `RelationArithmetic.jl` this unfortunately has to be done manually, but is often simple to do. This process could also be automated using an uncertainty precompiler [8].

### 5.2. Euler integration

In this example we propagate an interval through dynamic system governed by a non-linear ordinary differential equation (ODE). For this we implement a standard deterministic ODE solver (the Euler method) and perform interval dependence tracking through

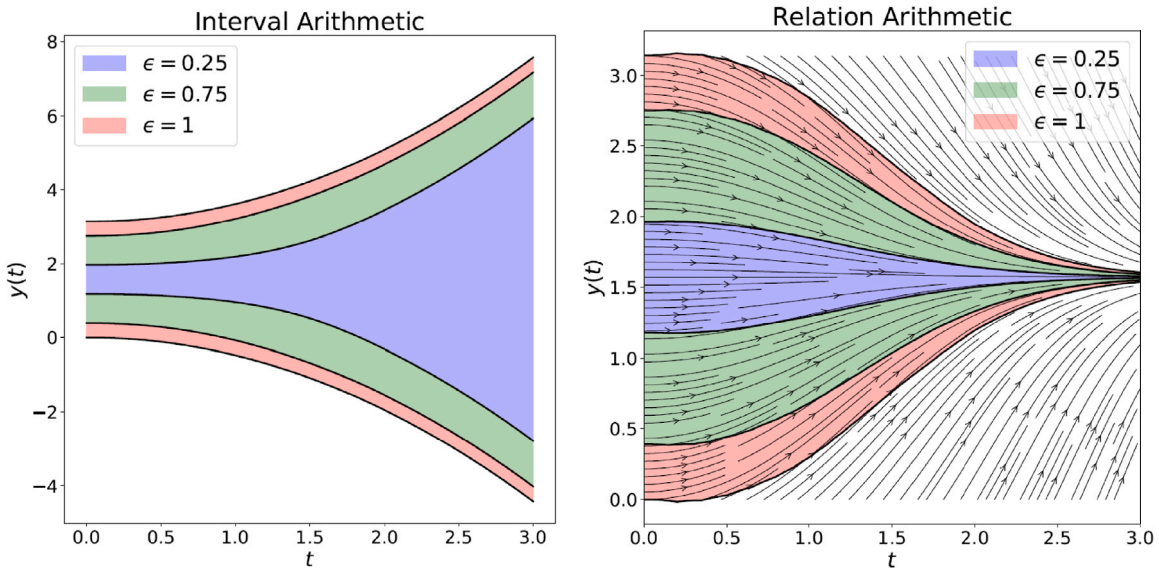


**Fig. 17.** Interval propagation through a non-linear ODE for different initial interval widths. Left shows standard interval arithmetic, and right shows relation arithmetic. Plotted on the right is also the ODE's flow-field, which the interval bounds follow and converge when dependencies are tracked.

the algorithm. We stress that if one wants to rigorously propagate sets in ODEs there are specialised methods and software for doing so, namely Taylor Models [14], Reachability Analysis [13], and Interval Contractors [20]. However this example is interesting because we take fairly complicated deterministic algorithm, which is completely unaware about intervals, and propagate intervals tightly through it.

We will compute the sets of trajectories of the ODE $\dot{y}(t) = f(t, y) = t\cos(y)$ for different interval initial value problems $Y_0 = \frac{\pi}{2} \pm \epsilon\frac{\pi}{2}$, with $\epsilon \in [0, 1]$ (i.e. $\epsilon = 1$ is $Y_0 = [0, \pi]$, and $\epsilon = 0$ is $Y_0 = \pi/2$), and $t_0 = 0$, $t_{\text{final}} = 3$. For a specified time step $h$, the Euler method computes the trajectory $y(t)$ by starting from the initial condition $Y_0$ and time stepping the trajectory's value $Y_i \approx Y_{i-1} + hf(t_{i-1}, Y_{i-1})$ and time $t_i = t_{i-1} + h$, until $t_{\text{final}}$ is reached. If $h$ is small enough, the computed trajectory is accurate. Since $Y_{i-1}$ is an interval, the function evaluation $Y_{\text{step}} = hf(t_{i-1}, Y_{i-1})$ also produces an interval. Therefore for the time step $Y_{i-1} + Y_{\text{step}}$ to be tightly evaluated with interval arithmetic, the dependence of $Y_{i-1}$ and $Y_{\text{step}}$ must be computed, which the method presented here will do automatically.

Fig. 17 show the results of using interval arithmetic (left) and relation arithmetic (right) directly in the Euler method for the different initial value problems $\epsilon = \{0.25, 0.75, 1\}$. The bounds from interval arithmetic diverge, whilst the bounds from relation arithmetic converge. Plotted on the right is also the flow-field of the ODE, which actually converges towards the point $\pi/2$. The interval bounds from relation arithmetic correctly follow the flow-field and converge to this point.

We stress again that this example is not intended as rigorous calculation of the ODEs trajectories, but as showcase of automatically inserting uncertainty into a deterministic algorithm. The Euler method only approximates the correct solution, and so the calculated bounds will not be a rigorous enclosure of the trajectories. The intervals will however bound all the Monte Carlo trajectories of the same Euler method. The method also can retrieve the dependencies between the different time values, i.e. $J_{Y_i, Y_{i+1}}$.

## 6. Extensions

In this section we discuss some optimisations and generalisations that can be made to the method.

### 6.1. Making the method more efficient

For large computer programs, one can see that a large amount of memory would be required to store all the dependencies. Even a simple expression such as $\sin(XY)(XZ)$ resulted in quite a complex network, as seen in Fig. 13. In the worst case, for $N$ overall variables used in a program, not only inputs but including variables that are initialised during the execution, $N^2 \times n \times n$ bits are required to store the entire relation network. However, this is an extreme case where all variables require an interaction with each other. To make the storage more efficient one could use a sparse bit matrix, whereby only the indices containing 1 are stored, as opposed to the entire bit matrix. This would be particularly effective in storing dependencies with low uncertainty (are mostly 0). For the non-interactive case, where all elements are 1, one could use a single bit to indicate so, i.e. a $1 \times 1$ bit matrix containing a single 1.

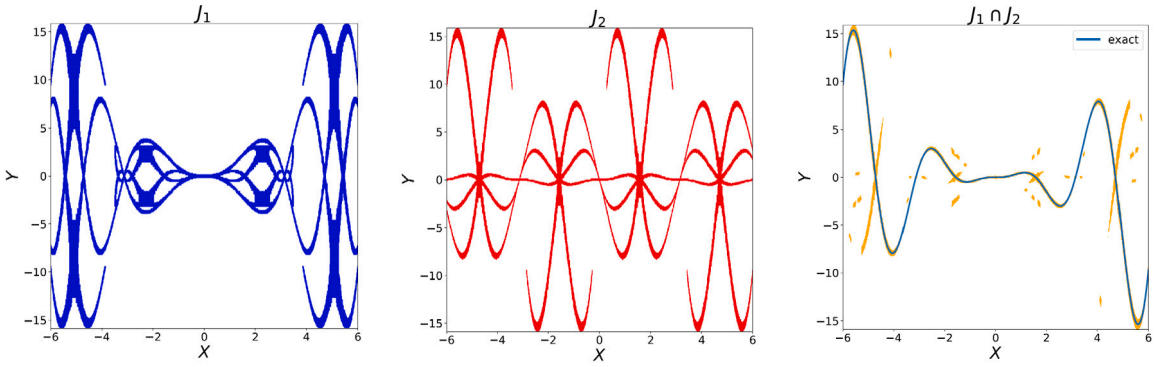


**Fig. 18.** Use of different relation network paths in the evaluation of $Y = X^2 \sin(X)\cos(X)$ to get a tighter dependence. $J_1$ and $J_2$ are intersected to get the right plot.

A further optimisation would be to not store every new relation after operations. Each binary operation creates two new dependencies ($J_{X\circ Y,X}$ and $J_{X\circ Y,Y}$) and each unary operation one new dependency ($J_{f(X),X}$). These dependencies may not be required in the calculation, for example $J_{W_1X}$ is unused in Fig. 13. As opposed to adding these dependencies each time, the mathematical operator relating the variables could be stored, and used in the calculation of the appropriate dependency when, and if, it is required by the program. Note that this would not lead to a dramatic saving in computational time, but would reduce the overall memory required by the method.

The method is also readily parallelisable. The conditioning steps used in arithmetic and inference are independent of each other, and could be calculated concurrently. The process of using different paths in the network to get the same dependence could also be parallelised fairly easily, leading to tighter dependencies and marginals.

### 6.2. Intersecting different paths

There are often not singular solutions to the chained dependence inferences from Section 4.2. We have seen that often multiple paths link variables, and that the order in which chained inference is performed can be selected. All solutions are however rigorous, and we can therefore combine these different rigorous calculations to get a tighter result. Take for example the calculation of $Y = X^2 \sin(X)\cos(X)$ for $X = [-6, 6]$. Fig. 18 shows two different calculations of the set $J_{YX}$, $J_1$ in blue and $J_2$ in red. Since these are two different rigorous calculations of the same quantity, they can be combined with a simple set intersection, which is shown on as the orange dependence on the right of Fig. 18. Set intersections can be easily performed with bit-matrices as element-wise Boolean *and* operations (or multiplication). The resulting intersection is a much tighter bound on the exact solution. Although this is likely too intensive for serial calculation, these different calculations could be designated to different parallel processes.

The order in which the operations are executed will also influence results. For example, solving the propagation problem for $(X^2 \sin(X))\cos(X)$ and $X^2(\sin(X)\cos(X))$ will give two different outer approximations to $J_{YX}$. In this work, no suggestion is given about the most efficient (tightest) order to execute. However like the above example, both approximations are rigorous and can be intersected to give a tighter result.

### 6.3. Tracking in higher dimensions

The presented method incorporates bivariate dependence into interval arithmetic, and as such gives outer approximations to exact multivariate dependencies. The method therefore may never contract to the exact interval bounds, even as higher discretisations are used. A generalisation would be to track trivariate or higher dependencies, which would lead to further tighter dependencies and marginals, and eventually the exact multivariate set of interest. Although this would introduce additional computational cost, likely to be similar or exactly the same as subintervalisation for large systems.

### 6.4. Copulas and p-boxes

Rigorous p-box calculations can be performed with p-box arithmetic or probability bounds analysis [6]. P-box arithmetic is based on convolutions involving copulas, i.e. to perform binary operations exactly a copula between the inputs is required, and as such also suffers from the repeated variable problem. A p-box generalisation of the presented method would be to calculate the copulas between inputs and outputs of operations, and use them for variables which are repeated in a program. This has been suggested by the authors [30], and they show that this is feasible to some extent, with some initial results. However, the generalisation of the proposed dependency network is not straightforward. Indeed it could be used to store copulas, however it is unlikely that inferences can be performed the same way as was proposed here. Unlike for intervals, two bivariate copulas $C_{XY}, C_{YZ}$ give little information

about the third $C_{XZ}$, and so likely a slightly different strategy is required for dependency inferences. The work of Durante et al. [31] on $\star$-products for copulas may be useful here. However, perhaps the method proposed here could be used to constrain the ranges of repeated bivariate p-boxes, and this in turn could be used to gain some information about the copula, for example where it has zero probability measure.

## 7. Conclusions

A novel method to incorporate dependence into interval arithmetic has been introduced. The presented method preserves the rigour for interval arithmetic, but additionally allows for dependencies between intervals to be track through a computational model's execution. It has been shown that when dependence is tracked this way, artificial uncertainty from repeated variables can be reduced. We represent bivariate interval relations as $n \times n$ bit-matrices, which serve as interval equivalents to copulas from probability theory. We show how dependencies can be tracked through binary and unary operations efficiently using this representation.

A central component and novelty of the presented method are interval relation networks, an interval analogue to vine copulas. In these networks, bivariate dependencies between variables are stored and used to infer any unknown dependence required in the calculation sequence, or indeed may be used to bound a higher dimensional interval dependence. It has been shown how to evolve a relation network as a calculation sequence is executed, and we present the theoretical and computation details about how inferences are performed. Unlike vine copulas, interval relation networks require no additional assumptions to compute inferences, such as specific network topologies or conditional independence. The interval relation network may be interesting in and of itself. Vine copulas have become a popular tool for modelling complex high dimensional probabilistic dependencies, and analysts presented with imprecise probabilistic information may find them useful.

The presented method is automatic, in the sense that intervals and relations can be specified at the beginning of one's scripts and any proceeding code will be tightly evaluated with arithmetic, including any inferences, allowing for the complexities of the method to be hidden away from the computational scientist. This has been demonstrated in an open source Julia package `RelationArithmetic.jl`, which we have applied to several engineering dynamics problems. The presented method may be used stand-alone, or may be incorporated into a larger uncertainty compiler or language framework [8], whereby additional syntactic tricks may be used to further reduce the effect of repeated variables. However, how the presented method can interact with p-box [6], possibilistic [7] and moment arithmetic [32] is an interesting theoretical question that would have to be addressed.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The code will be made available on GitHub: https://github.com/AnderGray/RelationArithmetic.jl

## Acknowledgements

We thank Enrique Miralles-Dolz and Dominic Calleja (both Risk Institute, University of Liverpool) for their useful comments after reading this paper. We also thank Vladik Kreinovich (University of Texas at El Paso) and the anonymous reviewers of this paper, who through comments, questions, and suggestions improved its quality.

The authors would like to thank the support from the EPSRC, United Kingdom iCase studentship award 15220067. We also gratefully acknowledge funding from UKRI via the EPSRC and ESRC Centre for Doctoral Training in Risk and Uncertainty Quantification and Management in Complex Systems. This research is funded by the Engineering & Physical Sciences Research Council (EPSRC), United Kingdom with grant no. EP/R006768/1, which is greatly acknowledged for its funding and support. This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014–2018 and 2019–2020 under grant agreement No. 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

## Appendix. Code for relation inference

The following is a Julia function for performing interval relation inference using bit-matrices, as described in Section 4.1.

```
function project(Rxy :: BitMatrix, Ryz :: BitMatrix)
  ###
  #  Returns projection Rxz from projections Rxy & Ryz
  #  Assumes size(Rxy, 2) == size(Ryz, 1)
  ###
  nx = size(Rxy, 1); nz = size(Ryz, 2);
  ny = size(Rxy, 2);

  Rxz = falses(nx, nz); # BitMatrix of falses

  for k = 1:ny
    is = findall(Rxy[:,k] .== 1)  # find conditionals
    js = findall(Ryz[k,:] .== 1)  # indices of trues

    [Rxz[i,j] = 1 for i in is, j in js] # Cartesian product
  end

  return Rxz
end
```

## References


[1] Ramon E. Moore, R. Baker Kearfott, Michael J. Cloud, Introduction to Interval Analysis, SIAM, 2009.
[2] Siegfried M. Rump, INTLAB—interval laboratory, in: Developments in Reliable Computing, Springer, 1999, pp. 77–104.
[3] Warwick Tucker, Validated Numerics, Princeton University Press, 2011.
[4] Robert Charles Williamson, et al., Probabilistic arithmetic (Ph.D. thesis), University of Queensland Australia, 1989.
[5] Scott Ferson, Janos G. Hajagos, Arithmetic with uncertain numbers: rigorous and (often) best possible answers, Reliab. Eng. Syst. Saf. 85 (1–3) (2004) 135–152.
[6] Ander Gray, Scott Ferson, Edoardo Patelli, Probabilityboundsanalysis.jl: Arithmetic with sets of distributions, Submitt. Proc. JuliaCon (2021).
[7] Ander Gray, Dominik Hose, Marco De Angelis, Michael Hanss, Scott Ferson, Dependent possibilistic arithmetic using copulas, in: International Symposium on Imprecise Probability: Theories and Applications, PMLR, 2021, pp. 169–179.
[8] Nicholas Gray, Marco De Angelis, Scott Ferson, The creation of puffin, the automatic uncertainty compiler, 2021, ArXiv E-Prints, arXiv–2110.
[9] Jeff Bezanson, Stefan Karpinski, Viral B Shah, Alan Edelman, Julia: A fast dynamic language for technical computing, 2012, arXiv preprint arXiv:1209.5145.
[10] Ramon E. Moore, Interval Analysis, Vol. 4, Prentice-Hall Englewood Cliffs, 1966.
[11] James M. Hyman, Forsig: an extension of fortran with significance arithmetic, Technical report, Los Alamos National Lab., NM (USA), 1982.
[12] Siegfried M. Rump, Masahide Kashiwagi, Implementation and improvements of affine arithmetic, Nonlinear Theory Appl. IEICE 6 (3) (2015) 341–359.
[13] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, Christian Schilling, JuliaReach: a toolbox for set-based reachability, in: HSCC '19: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, Apr 2019, Montreal, Canada, 2019, URL https://hal.archives-ouvertes.fr/hal-02446216.
[14] Kyoko Makino, Rigorous Analysis of Nonlinear Motion in Particle Accelerators, Michigan State University, 1998.
[15] Weiye Li, James Mac Hyman, Computer arithmetic for probability distribution variables, Reliab. Eng. Syst. Saf. 85 (1–3) (2004) 191–209.
[16] Vladik Kreinovich, Anatoly V Lakeyev, Jiří Rohn, Patrick Kahl, Computational complexity and feasibility of data processing and interval computations, Vol. 10, Springer Science & Business Media, 2013.
[17] Marco de Angelis, Exact bounds on the amplitude and phase of the interval discrete Fourier transform in polynomial time, arXiv (2022) http://dx.doi.org/10.48550/ARXIV.2205.13978.
[18] Eldon Hansen, G. William Walster, Global Optimization using Interval Analysis: Revised and Expanded, Vol. 264, CRC Press, 2003.
[19] Luc Jaulin, Michel Kieffer, Olivier Didrit, Eric Walter, Interval analysis, in: Applied Interval Analysis, Springer, 2001, pp. 11–43.
[20] Simon Rohou, Abderahmane Bedouhene, Gilles Chabert, Alexandre Goldsztejn, Luc Jaulin, Bertrand Neveu, Victor Reyes, Gilles Trombettoni, Towards a generic interval solver for differential-algebraic CSP, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2020, pp. 548–565.
[21] Scott Ferson, Vladik Kreinovich, Modeling correlation and dependence among intervals, in: Proceedings of the Second International Workshop on Reliable Engineering Computing, REC2006, 2006.
[22] Martine Ceberio, Scott Ferson, Vladik Kreinovich, Sanjeev Chopra, Gang Xiang, Adrian Murguia, Jorge Santillan, How to take into account dependence between the inputs: from interval computations to constraint-related set computations, with potential applications to nuclear safety, bio-and geosciences, in: Proceedings of the Second International Workshop on Reliable Engineering Computing, REC2006, 2006.
[23] Lech Polkowski, Rough Sets, Springer, 2002.
[24] Hermann Schichl, Ferenc Domes, Tiago Montanher, Kevin Kofler, Interval unions, BIT Numer. Math. 57 (2) (2017) 531–556.
[25] Harry Joe, Dorota Kurowicka, Dependence Modeling: Vine Copula Handbook, World Scientific, 2011.
[26] Scott Ferson, Janos Hajagos, Daniel Berleant, Jianzhong Zhang, W Troy Tucker, Lev Ginzburg, William Oberkampf, Dependence in Dempster-Shafer theory and probability bounds analysis, Sandia Natl. Lab. (2004).
[27] David P. Sanders, Luis Benet, et al., JuliaIntervals/IntervalArithmetic.jl: v0.19.2, Zenodo, 2021, http://dx.doi.org/10.5281/zenodo.5519761.
[28] James Fairbanks, Mathieu Besançon, Simon Schölly, Júlio Hoffiman, Nick Eubank, Stefan Karpinski, JuliaGraphs/Graphs.jl: an optimized graphs package for the Julia programming language, 2021, URL https://github.com/JuliaGraphs/Graphs.jl/.
[29] Seth Bromberger, Kevin Bonham, Mathieu Besançon, Simon Schölly, Stephan Kleinbölting, JuliaGraphs/MetaGraphs.jl: graph data structures with multiple heterogeneous metadata for graphs.jl., 2021, URL https://github.com/JuliaGraphs/MetaGraphs.jl.
[30] Ander Gray, Marco De Angelis, Scott Ferson, Edoardo Patelli, What's $Z - X$, when $Z = X + Y$? Dependency tracking in interval arithmetic with bivariate sets, in: Proceedings of the 9th International Workshop on Reliable Engineering Computing, REC2021, 2021.
[31] Fabrizio Durante, Erich Peter Klement, José Quesada-Molina, Peter Sarkoci, Remarks on two product-like constructions for copulas, Kybernetika 43 (2) (2007) 235–244.
[32] Ander Gray, Scott Ferson, Vladik Kreinovich, Edoardo Patelli, Distribution-free risk analysis, Internat. J. Approx. Reason. (2022).