Advanced Math Improvements in Maple 2025

Maple 2025 includes many improvements to the math engine.

PartiallyOrderedSets

- Maple 2025 contains a new package for computing with partially ordered sets, also known as posets. After considering a random poset, we illustrate the capabilities of this package with four examples taken from different fields of mathematics.

```
> with(PartiallyOrderedSets):
```

- A partially ordered set is a set equipped with a relation that is reflexive, antisymmetric, and transitive. For this package, only finite sets are considered. The relation is usually denoted by the $\leq$ relator.

A random poset

- Posets can be constructed in various ways. One way is to supply an adjacency matrix. Below, we generate a random matrix that is valid as an adjacency matrix for a poset: we first generate a fully random 0-1 matrix with appropriate density, set its lower triangle to 0 and its diagonal to 1, and then find the transitive closure.

```
> n := 25:
```

```
> L := LinearAlgebra:-RandomMatrix(n, generator=1, density=1/3);
```

$$L := \begin{bmatrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & \dots & \cdots \\ 2 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & \dots & \cdots \\ 3 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & \dots & \cdots \\ 4 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & \dots & \cdots \\ 5 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & \dots & \cdots \\ 6 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \dots & \cdots \\ 7 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \dots & \cdots \\ 8 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & \dots & \cdots \\ 9 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & \dots & \cdots \\ 10 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & \dots & \cdots \\ & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

25 × 25 Matrix

```
> L := ArrayTools:-UpperTriangle(L):
> L := max~(L, LinearAlgebra:-IdentityMatrix(n)):
> L := min~(1, L^n);
```

$$L := \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ \\ \end{array} \begin{bmatrix} & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & \\ \cdots & 0 & 0 & 1 & \ldots & & & & & & \cdots \\ \cdots & 1 & 1 & 1 & \ldots & & & & & & \cdots \\ \cdots & 0 & 1 & 1 & \ldots & & & & & & \cdots \\ \cdots & 0 & 0 & 1 & \ldots & & & & & & \cdots \\ \cdots & 0 & 0 & 1 & \ldots & & & & & & \cdots \\ \cdots & 0 & 0 & 1 & \ldots & & & & & & \cdots \\ \cdots & 1 & 0 & 0 & \ldots & & & & & & \cdots \\ \cdots & 0 & 1 & 0 & \ldots & & & & & & \cdots \\ \cdots & 0 & 0 & 1 & \ldots & & & & & & \cdots \\ \cdots & 0 & 0 & 0 & \ldots & & & & & & \cdots \\ & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

25 × 25 Matrix

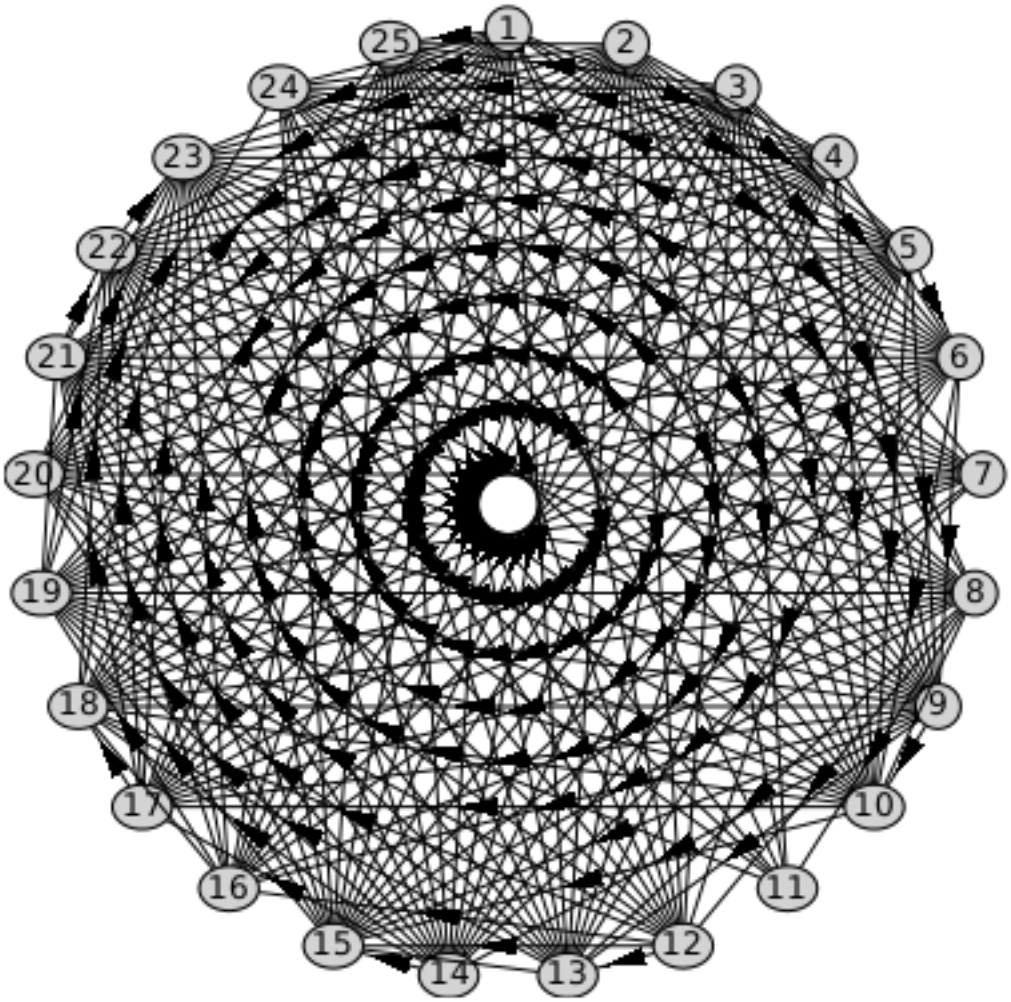
- Now we generate the poset. We specify the base set (in this case, the integers 1 through n) and the relation (defined by L).

```
> random_poset := PartiallyOrderedSet([seq(1 .. n)], L);
```

$$random\_poset := \langle \text{a poset with 25 elements} \rangle$$

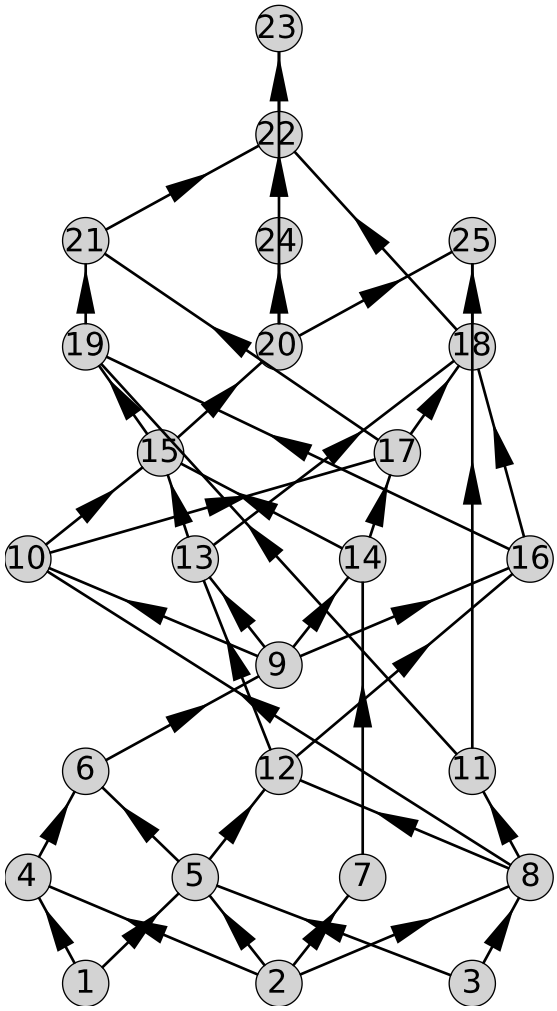
- We can visualize this poset in multiple ways. The naive way would be to draw an arrow between every pair of related elements, as follows.

```
> DrawGraph(random_poset, reduction = false);
```



- This does not give much insight. A better approach is showing the so-called Hasse diagram, where we only show an arrow from a to c if $a < c$ and there is no b such that $a < b < c$.

```
> DrawGraph(random_poset);
```



The poset of the divisors of a positive integer

• We define the relation of divisibility as a procedure, and create the poset on the divisors of 2025 using this procedure.

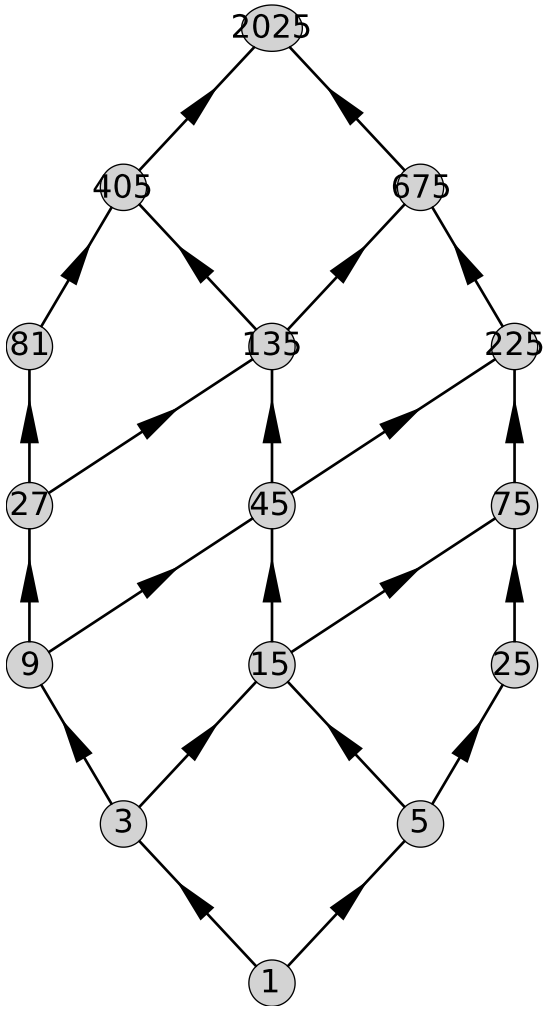
```
> divisibility := (x, y) -> irem(y, x) = 0:
```

```
> divisibility_poset := PartiallyOrderedSet(NumberTheory:-Divisors
  (2025), divisibility);
```

$$divisibility\_poset := \langle \text{a poset with 15 elements} \rangle$$

• We display this poset.

```
> DrawGraph(divisibility_poset);
```



- There are a number of properties that a poset may or may not have that we can test using this package. A poset is graded if its elements can be partitioned into numbered subsets such that the immediate successors of the entries of subset n are numbered $n+1$.

```
> IsGraded(divisibility_poset);
```

true

- A poset is ranked if its maximal chains (that is, subsets where every element is related to every other element) all have the same cardinality. Every ranked poset is graded, but this is not true the other way around.

```
> IsRanked(divisibility_poset);
```

true

- A poset is a lattice if every pair of elements has a smallest upper bound and a greatest lower bound.

```
> IsLattice(divisibility_poset);
```

true

- A poset is a face lattice if it is isomorphic to the face lattice of a polyhedral set (see below).

```
> IsFaceLattice(divisibility_poset);
```

$$true$$

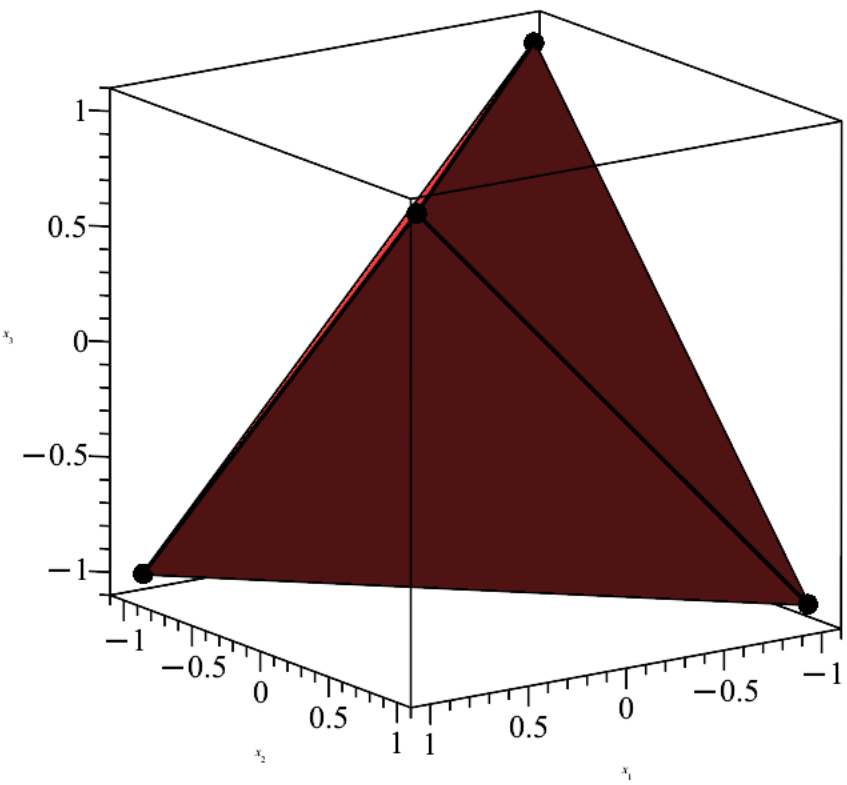
The face lattice of a polyhedral set

- A polyhedral set is the intersection of a finite number of half spaces. An example is a tetrahedron.

```
> t := PolyhedralSets:-ExampleSets:-Tetrahedron();
```

$$t := \begin{cases} Coordinates & : & [x_1, x_2, x_3] \\ Relations & : & [-x_1 - x_2 - x_3 \le 1, -x_1 + x_2 + x_3 \le 1, x_1 - x_2 + x_3 \le 1, x_1 + x_2 - x_3 \le 1] \end{cases}$$

```
> PolyhedralSets:-Plot(t);
```




```
> d := PolyhedralSets:-Dimension(t);
```

$$d := 3$$

- The faces of a polyhedral sets are the vertices, edges, and higher-dimensional boundaries of the polyhedral set. We sometimes also include the empty subset.

```
> t_faces := {seq(op(PolyhedralSets:-Faces(t, dimension = i)), i = 0
  .. d), PolyhedralSets:-ExampleSets:-EmptySet(d)}:
```

```
> face_list := convert(t_faces, list):
```

- We now construct the face lattice, ordered by inclusion.

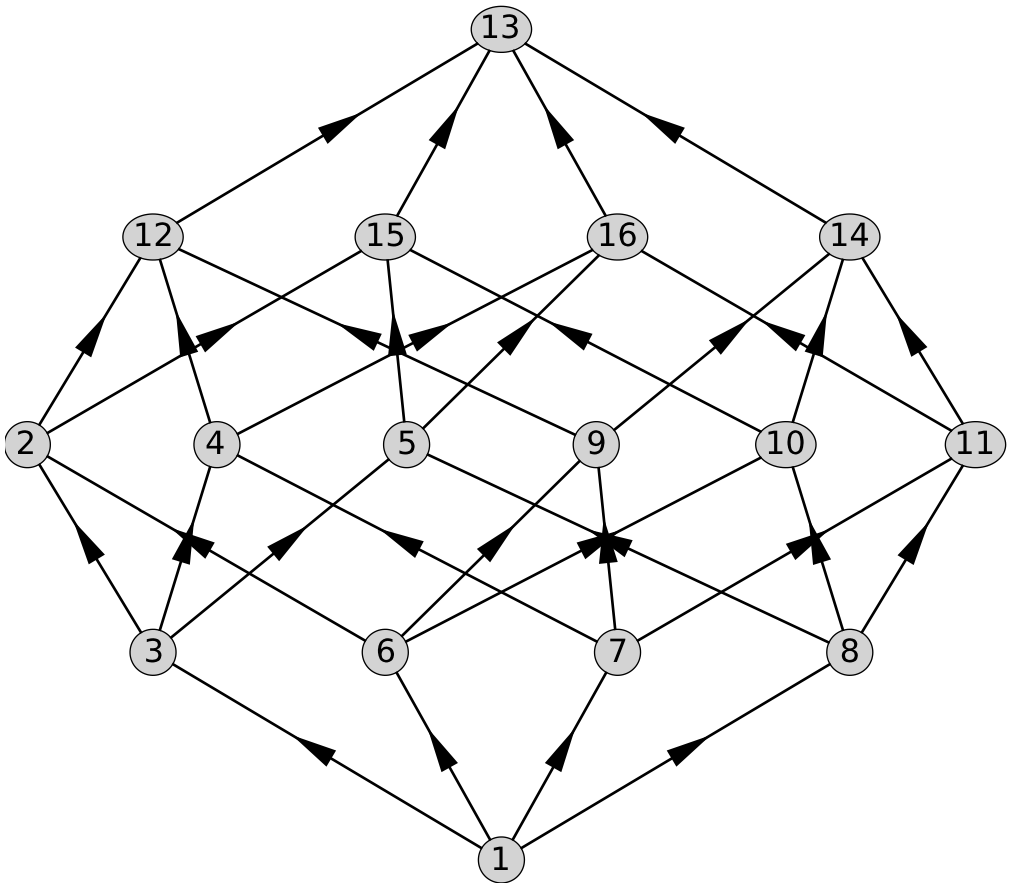
```
> inclusion := (x, y) -> PolyhedralSets:-`subset`(face_list[x],
  face_list[y]);
```

$$inclusion := (x, y) \mapsto PolyhedralSets:-subset(face\_list_x, face\_list_y)$$

```
> polyhedral_poset := PartiallyOrderedSet({seq(1 .. numelems
  (face_list))}, inclusion);
```

$$polyhedral\_poset := \langle a\ poset\ with\ 16\ elements \rangle$$

```
> DrawGraph(polyhedral_poset);
```



• We verify some properties of this poset.

```
> IsGraded(polyhedral_poset);
```

true

```
> IsRanked(polyhedral_poset);
```

true

```
> IsLattice(polyhedral_poset);
```

true

```
> IsFaceLattice(polyhedral_poset);
```

true

• The width of a poset is the size of the largest antichain (a set where every pair of elements is *not* related).

```
> Width(polyhedral_poset);
```

• The height of a poset is the size of the largest chain.

```
> Height(polyhedral_poset);
```

$$5$$

The lattice of the flats of a matroid

• Consider the Fano matroid. Its flats form a partially ordered set, ordered by inclusion. We ensure compact display of the flats with some custom procedures.

```
> M := Matroids:-ExampleMatroids:-Fano();
```

$$M := \langle \text{a matroid on 7 elements with 14 circuits} \rangle$$

```
> F := Matroids:-Flats(M);
```

$$F := [\varnothing, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{1, 2, 4\}, \{1, 3, 5\}, \{2, 3, 6\}, \{4, 5, 6\}, \{3, 4, 7\}, \{2, 5, 7\}, \{1, 6, 7\}, \{1, 2, 3, 4, 5, 6, 7\}]$$

```
> compact_display := proc(s :: set, $)
  if s = {} then
    return "&empty;";
  else
    return String(seq(s));
  end if;
  end proc:
```

```
> Compact_F := map(compact_display, F);
```

$$Compact\_F := [\text{"}\varnothing\text{"}, \text{"1"}, \text{"2"}, \text{"3"}, \text{"4"}, \text{"5"}, \text{"6"}, \text{"7"}, \text{"124"}, \text{"135"}, \text{"236"}, \text{"456"}, \text{"347"}, \text{"257"}, \text{"167"}, \text{"1234567"}]$$

```
> Expand_F := table(zip(`=`, Compact_F, F));
```

$$Expand\_F := \text{table}([\text{"4"} = \{4\}, \text{"2"} = \{2\}, \text{"1234567"} = \{1, 2, 3, 4, 5, 6, 7\}, \text{"1"} = \{1\}, \text{"236"} = \{2, 3, 6\}, \text{"456"} = \{4, 5, 6\}, \text{"7"} = \{7\}, \text{"3"} = \{3\}, \text{"}\varnothing\text{"} = \varnothing, \text{"6"} = \{6\}, \text{"347"} = \{3, 4, 7\}, \text{"167"} = \{1, 6, 7\}, \text{"124"} = \{1, 2, 4\}, \text{"135"} = \{1, 3, 5\}, \text{"5"} = \{5\}, \text{"257"} = \{2, 5, 7\}])$$

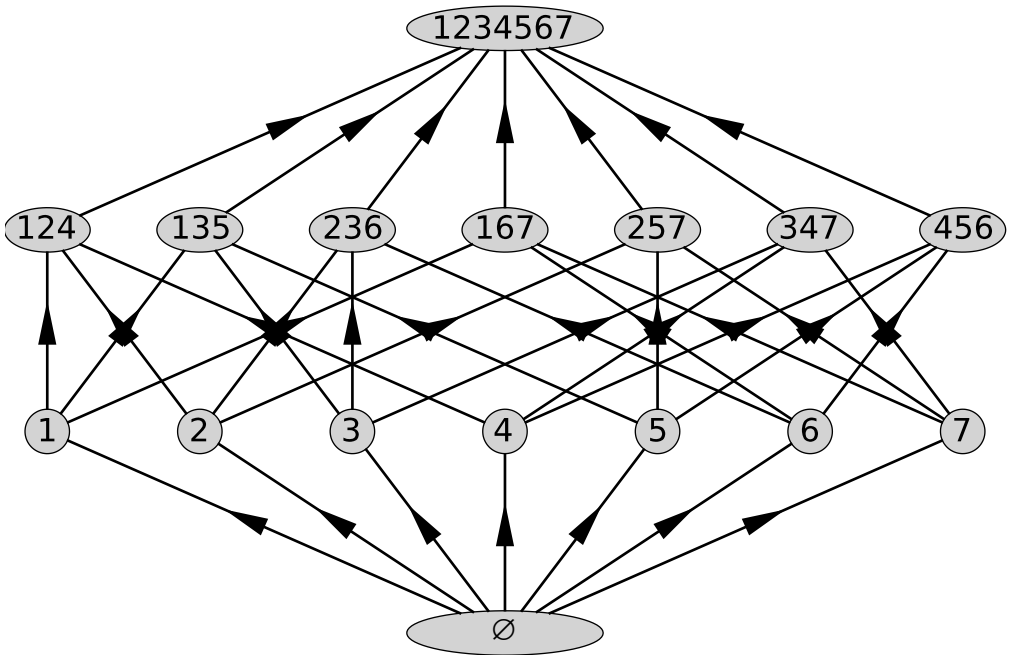
```
> inclusion := (x, y) -> Expand_F[x] subset Expand_F[y];
```

$$inclusion := (x, y) \mapsto Expand\_F_x \subseteq Expand\_F_y$$

```
> matroid_poset := PartiallyOrderedSet({op(Compact_F)}, inclusion);
```

$$matroid\_poset := \langle \text{a poset with 16 elements} \rangle$$

```
> DrawGraph(matroid_poset);
```



• Observe that this is the lattice of the faces of a polyhedral set, but not of the tetrahedron.

```
> IsLattice(matroid_poset);
```

true

```
> IsFaceLattice(matroid_poset);
```

true

```
> AreIsomorphic(polyhedral_poset, matroid_poset);
```

false

The poset of the partitions of a set

• The partitions of a set are partially ordered by the "finer-than" relation. One partition X is finer than another partition Y if each set in X is a subset of a set in Y.

```
> P := combinat:-setpartition({1,2,3,4});
```

$P := \{\{\{1,2,3,4\}\}, \{\{1\},\{2,3,4\}\}, \{\{2\},\{1,3,4\}\}, \{\{3\},\{1,2,4\}\}, \{\{4\},\{1,2,3\}\}, \{\{1,2\},$

$\{3,4\}\},\ \{\{1,3\},\{2,4\}\},\ \{\{1,4\},\{2,3\}\},\ \{\{1\},\{2\},\{3,4\}\},\ \{\{1\},\{3\},\{2,4\}\},\ \{\{1\},\{4\},$
$\{2,3\}\},\ \{\{2\},\{3\},\{1,4\}\},\ \{\{2\},\{4\},\{1,3\}\},\ \{\{3\},\{4\},\{1,2\}\},\ \{\{1\},\{2\},\{3\},\{4\}\}\}$

```
> IsFiner := proc(X :: set(set), Y :: set(set), $)
  for local x in X do
    if not ormap(y -> x subset y, Y) then
      return false;
    end if;
  end do;
  return true;
  end proc:
```

- Again we ensure compact display with some custom procedures.

```
> compact_display_2 := (s :: set(set)) -> StringTools:-Join(map
  (compact_display, [op(s)]), "|");
```

$$compact\_display\_2 := s::set(set) \mapsto StringTools:-Join(map(compact\_display, [op(s)]), "|")$$

```
> compact_P := map(compact_display_2, [op(P)]);
```

$compact\_P := [$"1234", "1|234", "2|134", "3|124", "4|123", "12|34", "13|24", "14|23", "1|2|34", "1|3|24", "1|4|23", "2|3|14", "2|4|13", "3|4|12", "1|2|3|4"$]$

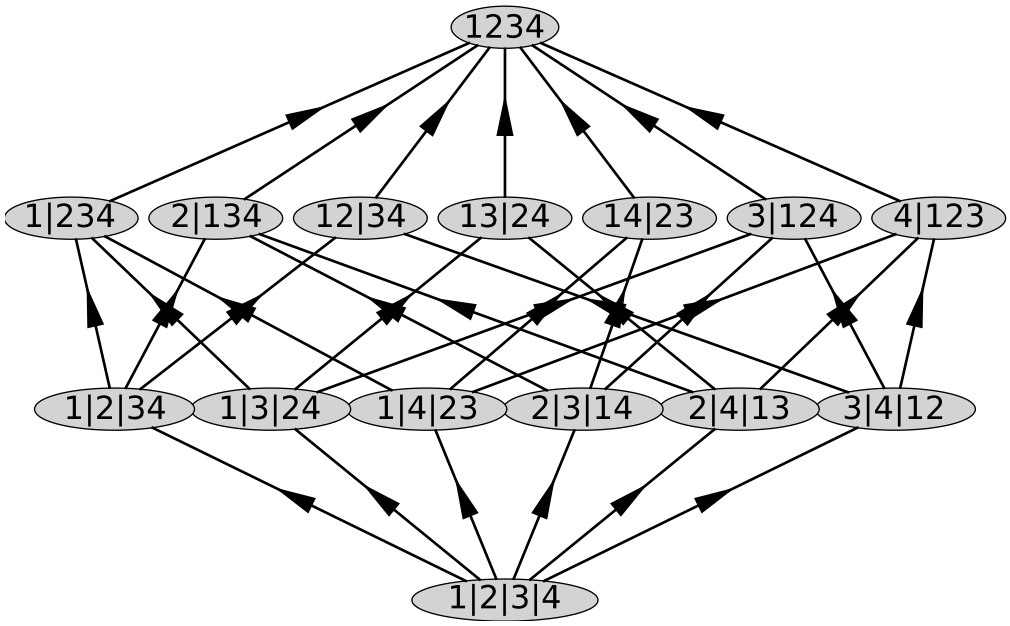
```
> expand_P := table(zip(`=`, compact_P, [op(P)])):

> compactIsFiner := (x, y) -> IsFiner(expand_P[x], expand_P[y]);
```

$$compactIsFiner := (x, y) \mapsto IsFiner(expand\_P_x, expand\_P_y)$$

```
> partition_poset := PartiallyOrderedSet(convert(compact_P, set),
  compactIsFiner);
```

$$partition\_poset := \langle a\ poset\ with\ 15\ elements \rangle$$

```
> DrawGraph(partition_poset);
```



• This is another face lattice.

```
> IsFaceLattice(partition_poset);
```

true

simplify

• Maple 2025 introduces several important improvements to simplify regarding expressions containing exponential, trigonometric, and/or inverse trigonometric functions. These updates also bring beneficial ripple effects across the Maple library.

New and improved conversion of exp to trig

• simplify now recognizes when exponentials can be profitably converted to hyperbolic trig functions:

```
> restart;
```

```
> simplify(exp(x)-exp(-x));
```

$$2\sinh(x)$$

```
> simplify((exp(x)-1)/(exp(x)+1));
```

$$\tanh\left(\frac{x}{2}\right)$$

```
> simplify(1/4/(1/2*exp(1)-1/2*exp(-1))*(exp(3)-exp(-3)-exp(1)+exp
  (-1)));
```

$$\cosh(2)$$

- Moreover, conversion to non-hyperbolic trig functions is now recognized in more cases than previously:

```
> simplify(I*(exp(2*I*x*y)+exp(-2*I*x*(-2+y)))/(exp(4*I*x)-1));
```

$$\cos(2\,x\,(-1+y))\,\csc(2\,x)$$

Simplification of expressions with exp or trig is more careful

- In general, simplification w.r.t. exp is now more careful to avoid unnecessary expansions and normalizations. This result is now significantly more compact:

```
> simplify(1/2*(-sin(t-x)*exp(I*1/6*abs(t-x))-cos(3*t-3*x))*exp(-
  I/6*abs(t-x)));
```

$$-\frac{e^{-\frac{I}{6}|t-x|}\cos(3\,t-3\,x)}{2}-\frac{\sin(t-x)}{2}$$

- In certain cases involving trigonometric functions, simplify now tries harder to get a fully simplified answer before returning:

```
> simplify(-sin(2)^2/2 - cos(2)^2/2 - sin(2)*cos(x - 1)*sin(x - 1) +
  cos(2)*cos(2*x - 2)/2);
```

$$\frac{\cos(2\,x)}{2}-\frac{1}{2}$$

Improved simplification of expressions with inverse trigonometric functions

- Trigonometric functions with logarithms or inverse trig functions in their arguments are now converted to a simpler form without trig and arctrig/ln when it can be determined that it leads to a simpler result:

```
> simplify(cos(ln(2)+arctanh(x))-2^(-1+I)*(1-x)^(-1/2*I)*(x+1)^(1/2*
  I));
```

$$2^{-1-\mathrm{I}}\left(1-x\right)^{\frac{\mathrm{I}}{2}}\left(1+x\right)^{-\frac{\mathrm{I}}{2}}$$

- simplify also now recognizes when converting arctan or arctanh to ln reduces the size of the expression:

```
> simplify(2*arctanh(x) + ln(1 - x));
```

$$\ln\left(1+x\right)$$

Conversion of exp(ln(t)) and exp(Pi*I*t)

- A trivial simplification was previously missed (it should be noted, this input is rare due to the uneval quotes; without them exp itself would make this simplification):

```
> simplify('exp'(ln(t)));
```

$$t$$

- While simplify does not by default convert exp(Pi*I*t) to a power for symbolic t, it will now do so if it detects that the resulting power already exists in the expression:

```
> simplify((1+(-1)^t)/exp(Pi*I*t));
```

$$\left(-1\right)^{-t}+1$$

Resulting improvements to other library commands

- The aforementioned improvements to simplify have had knock-on effects in other areas of the Maple library. For example, this limit is now computed correctly:

```
> q := exp(s*t)/s/cosh(s):
```

```
> p := (2*n+1)/2*Pi*I:
```

```
> limit((s-p)*q, s=p) assuming n::integer;
```

$$-\frac{2\,\mathrm{e}^{\frac{\mathrm{I}}{2}\left(2n+1\right)\pi t}\left(-1\right)^{n}}{\left(2n+1\right)\pi}$$

and these results are more compact:

```
> Student:-Calculus1:-SurfaceOfRevolution(cosh(x), x=-Pi..Pi, axis=
  vertical);
```

$$4\sinh\left(\pi\right)\pi^{2}-4\cosh\left(\pi\right)\pi+4\pi$$

```
> dsolve({diff(y(x),x,x)-2*y(x)=0, y(0)=1.2, y(1)=0.9});
```

$$y(x) = -\frac{3\left(-3\sinh\left(\sqrt{2}\,x\right) + 4\sinh\left(\sqrt{2}\,(x-1)\right)\right)\operatorname{csch}\left(\sqrt{2}\right)}{10}$$

```
> pdsolve([diff(u(x, y), x, x)+diff(u(x, y), y, y)-k*u(x, y) = 0, u
  (0, y) = 1, u(Pi, y) = 0, u(x, 0) = 0, u(x, Pi) = 0]) assuming k >
  0;
```

$$u(x,y) = \sum_{n=1}^{\infty}\left(-\frac{2\left((-1)^n - 1\right)\sin(n\,y)\sinh\left(\sqrt{n^2+k}\,(\pi - x)\right)\operatorname{csch}\left(\sqrt{n^2+k}\,\pi\right)}{\pi\,n}\right)$$

SumTools

- Two new commands were added to the SumTools[DefiniteSum] subpackage to test for convergence of an infinite series and find its convergence radius. See Radius of Convergence for details.

Numerical Inverse Laplace Transform

- Users on MaplePrimes have long requested an efficient method of numerically inverting a transfer function for which no symbolic inverse exists. In response, in Maple 2025 there is a new quadrature method for computing numerical inverse Laplace transforms, which is available by using the numeric option to the inttrans:-invlaplace command. The new calling sequence takes an expression or procedure and produces an Array of evaluation values of the inverse transform. See inttrans/invlaplace/numeric for details.

```
> expr := exp(0.3*s)*sinh(0.5*sqrt(s))/(s*(sqrt(s)*cosh(sqrt(s)) +
  s));
```

$$expr := \frac{e^{0.3\,s}\sinh\left(0.5\sqrt{s}\right)}{s\left(\sqrt{s}\cosh\left(\sqrt{s}\right) + s\right)}$$

- No symbolic inverse exists.

```
> inttrans:-invlaplace(expr, s, t);
```

$$\mathcal{L}^{-1}\left(\frac{e^{\frac{3}{10}\,s}\sinh\left(\frac{1}{2}\sqrt{s}\right)}{s^{3/2}\cosh\left(\sqrt{s}\right) + s^2}, s, t\right)$$

- Numerical inversion

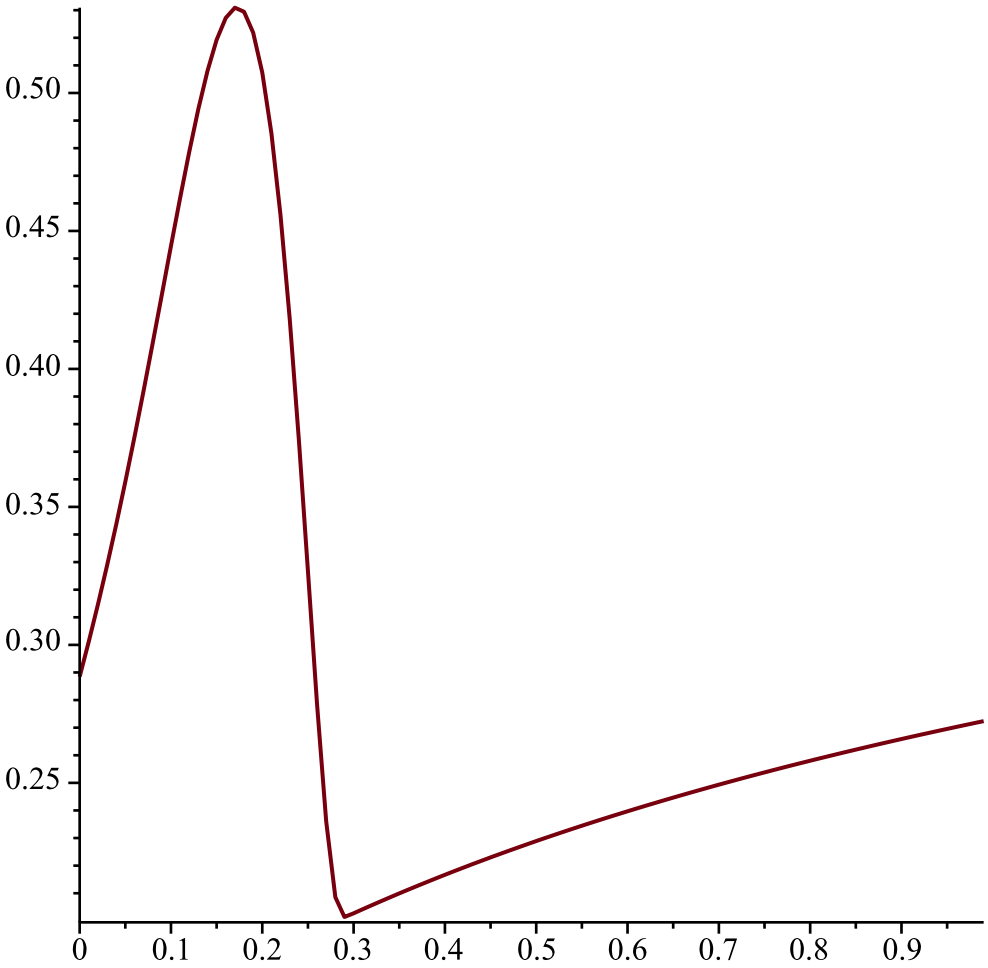
```
> numerical_inversion := inttrans:-invlaplace(expr, 100, 0.01,
  'numeric');
```

$$numerical\_inversion := \begin{bmatrix} 1 & 0.288496901896272 \\ 2 & 0.301224197497087 \\ 3 & 0.314634106413670 \\ 4 & 0.328749132382160 \\ 5 & 0.343577191735010 \\ 6 & 0.359106572391760 \\ 7 & 0.375299430630120 \\ 8 & 0.392083735616215 \\ 9 & 0.409343671113175 \\ 10 & 0.426908647709039 \\ & \vdots \end{bmatrix}$$

100 element Vector[column]

- Plot the simulated result.

```
> Times := Vector(100, i -> (i-1)*0.01):
> plot(Times, numerical_inversion);
```



Solver for complex functions

- The new command SolveTools:-ComplexSolve uses some transformations to produce better complex-valued solutions to equations involving abs $|z|$ and conjugate $\overline{z}$ than solve.

```
> SolveTools:-ComplexSolve( { abs(z)^2 + z*abs(z) + conjugate(z) },
  {z} );
```

$$\left[[z=0], \left[z=-\frac{1}{2}-\frac{I\sqrt{3}}{2}\right], \left[z=-\frac{1}{2}+\frac{I\sqrt{3}}{2}\right]\right]$$

PolyhedralSets

- In Maple 2025, the PolyhedralSets package has a number of new features, which we illustrate with the examples below.

```
> restart;
```

```
> with(PolyhedralSets):
```

```
> with(ExampleSets):
```

Computing the number of integer points for non-parametric polyhedral sets

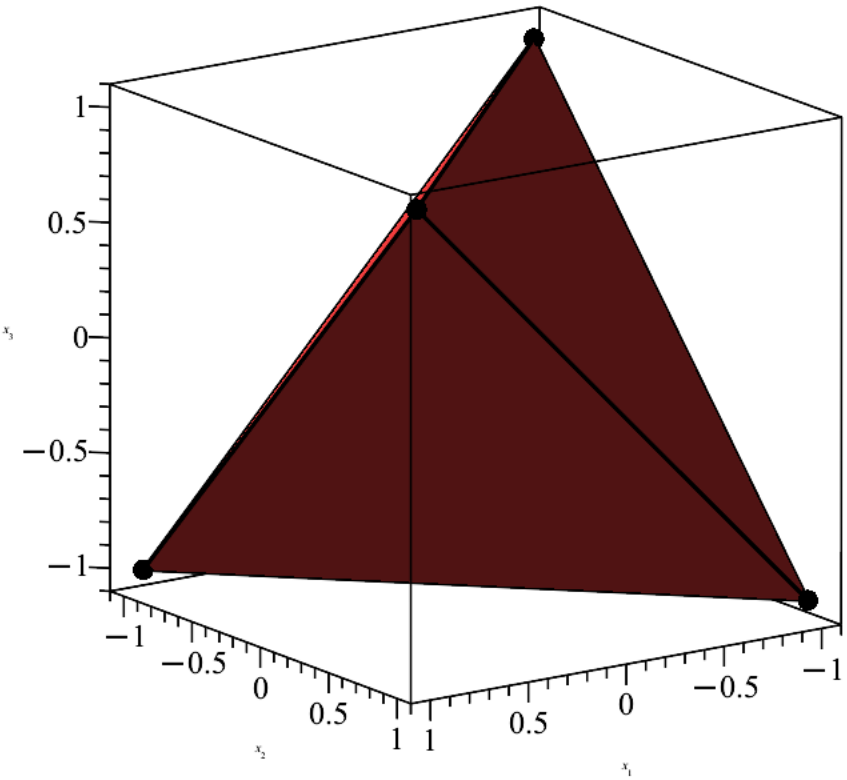
- The generating function of a polyhedral set in a space with coordinate variables $x_1,\ldots,x_n$ is the formal sum of the monomials $x_1^{a_1}\cdots x_n^{a_n}$, where $(a_1,\ldots,a_n)$ iterates over all integer points in the polyhedral set. Here is an example.

```
> ps := Tetrahedron();
```

$$ps :=$$

$$\begin{cases} Coordinates & : & [x_1, x_2, x_3] \\ Relations & : & [-x_1 - x_2 - x_3 \le 1, -x_1 + x_2 + x_3 \le 1, x_1 - x_2 + x_3 \le 1, x_1 + x_2 - x_3 \le 1] \end{cases}$$

```
> Plot(ps);
```



- From the plot, we can see that this polyhedral set has all its coordinates between -1 and 1. So we can enumerate all points in that box, and test if the relations defining the polyhedral set hold. This gives us the number of integer points.

```
> relations := Relations(ps);
```

$$relations := \left[-x_1 - x_2 - x_3 \le 1, -x_1 + x_2 + x_3 \le 1, x_1 - x_2 + x_3 \le 1, x_1 + x_2 - x_3 \le 1\right]$$

```
> candidates := [seq(seq(seq([i, j, k], i=-1..1), j=-1..1), k=-1..1)
  ];
```

$$candidates := [[-1,-1,-1],[0,-1,-1],[1,-1,-1],[-1,0,-1],[0,0,-1],[1,0,-1],[-1,1,-1],[0,1,-1],[1,1,-1],[-1,-1,0],[0,-1,0],[1,-1,0],[-1,0,0],[0,0,0],[1,0,0],[-1,1,0],[0,1,0],[1,1,0],[-1,-1,1],[0,-1,1],[1,-1,1],[-1,0,1],[0,0,1],[1,0,1],[-1,1,1],[0,1,1],[1,1,1]]$$

```
> points_inside := select(c -> andmap(r -> eval(r, [x[1], x[2], x[3]
  ] =~ c), relations), candidates);
```

$$points\_inside := [[1,-1,-1],[0,0,-1],[-1,1,-1],[0,-1,0],[-1,0,0],[0,0,0],[1,0,0],[0,1,0],[-1,-1,1],[0,0,1],[1,1,1]]$$

```
> numelems(points_inside);
```

$$11$$

- We can automate this process partially by computing the generating function, using the new GeneratingFunction command. The generating function has a term $x_1^{a_1} x_2^{a_2} x_3^{a_3}$ for every integer point (a_1, a_2, a_3) in the polyhedral set; for example, $\frac{x_2}{x_1 x_3}$ for the point $(-1, 1, -1)$.

```
> gps := GeneratingFunction(ps);
```

$$gps := \frac{x_1^2 x_2^2 x_3^2 + x_1^2 x_2 x_3 + x_3 x_1 x_2^2 + x_2 x_1 x_3^2 + x_1 x_2 x_3 + x_1^2 + x_1 x_2 + x_1 x_3 + x_2^2 + x_2 x_3 + x_3^2}{x_1 x_2 x_3}$$

- If we substitute 1 for each of the coordinate variables, this gives us the number of integer points.

```
> eval(gps, [x[1], x[2], x[3]] =~ 1);
```

$$11$$

- We can automate this even further by using the new NumberOfIntegerPoints command.

```
> NumberOfIntegerPoints(ps);
```

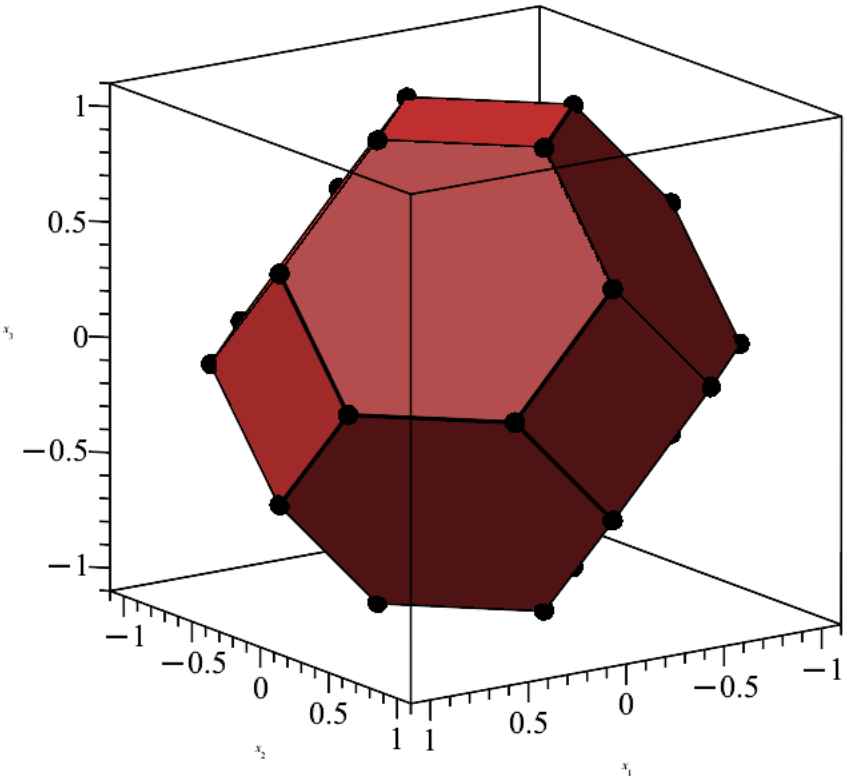
$$11$$

- Consider another rational polyhedral set.

```
> ps := TruncatedOctahedron();
```

$$ps := \begin{cases} Coordinates & : & [x_1, x_2, x_3] \\ Relations & : & \left[-x_3 \le 1, x_3 \le 1, -x_2 \le 1, x_2 \le 1, -x_1 - x_2 - x_3 \le \frac{3}{2}, -x_2 + x_3 - x_1 \le \frac{3}{2}, -x_1 \right. \end{cases}$$

```
> Plot(ps);
```




```
> NumberOfIntegerPoints(ps);
```

$$7$$

Computing the number of integer points for parametric polyhedral sets

- The NumberOfIntegerPoints command can also compute the number of integer points for *parametric* polyhedral sets, that is, a polyhedral set where the defining relations

include one or more parameters. This sort of analysis often occurs when analyzing nested loops in compiler optimization. For example, here is a right triangle with n points on each leg:

```
> parametric_triangle := [1 <= i, i <= n, 1 <= j, j <= i];
```

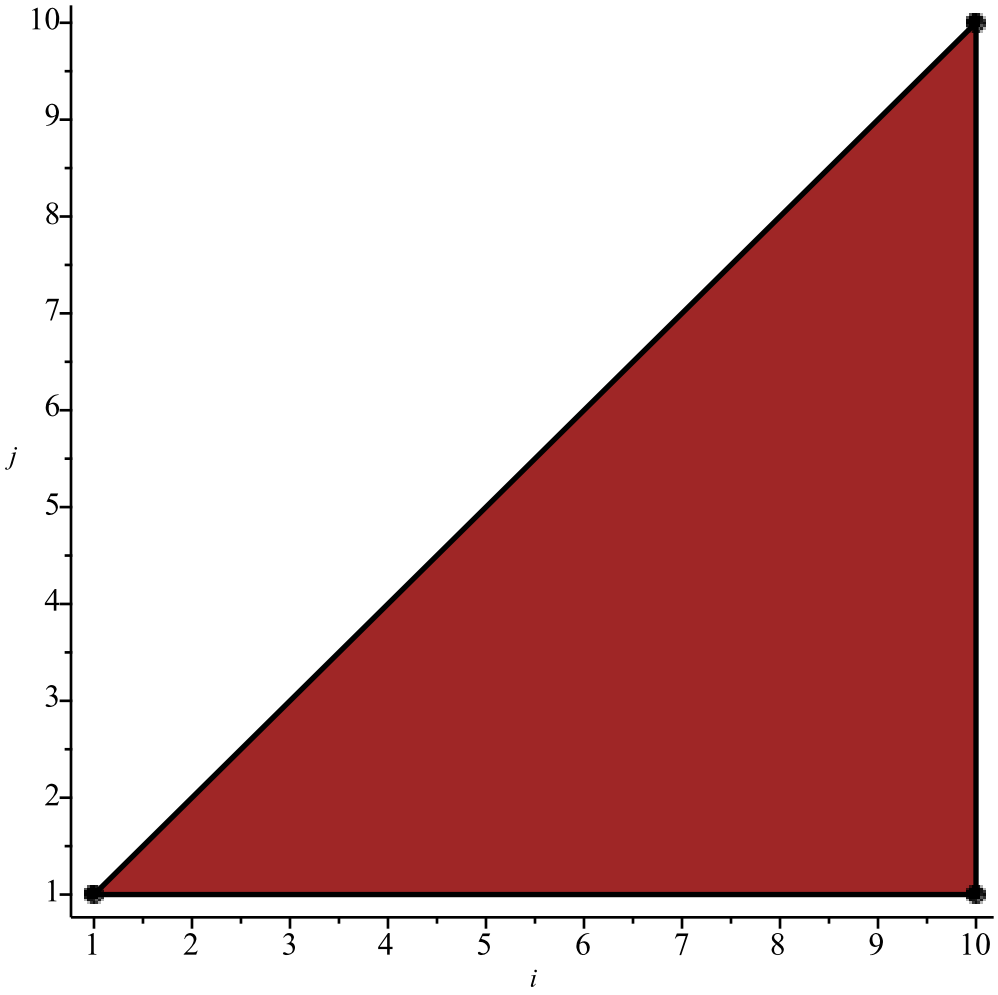
$$parametric\_triangle := [1 \le i, i \le n, 1 \le j, j \le i]$$

- We can plot a parametric triangle only if we specialize the parameters to a concrete value. Here we specialize to $n = 10$.

```
> ps := PolyhedralSet(eval(parametric_triangle, n=10));
```

$$ps := \begin{cases} Coordinates & : & [i, j] \\ Relations & : & [-j \le -1, -i + j \le 0, i \le 10] \end{cases}$$

```
> Plot(ps);
```



- We can of course compute the number of integer points of this specialized triangle.

```
> NumberOfIntegerPoints(ps);
```

- To compute the number of integer points of the parametric triangle, we need to tell Maple which of the variables are meant to be coordinate variables and which are meant to be parameters. When we do so, we get back a formula for the number of integer points in terms of the parameter, n.

```
> np := NumberOfIntegerPoints(parametric_triangle, [i, j], [n],
  output = piecewise);
```

$$np := \begin{cases} \left\{ \frac{1}{2} n^2 + \frac{1}{2} n \right\} & 0 \le -2 + n \\ \{1\} & n - 1 = 0 \end{cases}$$

```
> eval(np, n = 10);
```

$$\{55\}$$

- The following parametric triangle is more complicated, because the formula is different depending on whether n is even or odd.

```
> parametric_triangle_2 := [1 <= i, j <= n, 2*i <= 3*j];
```

$$parametric\_triangle\_2 := [1 \le i, j \le n, 2\,i \le 3\,j]$$

```
> ps_8 := PolyhedralSet(eval(parametric_triangle_2, n=8));
```

$$ps\_8 := \begin{cases} Coordinates & : & [i, j] \\ Relations & : & \left[j \le 8, -i \le -1, i - \frac{3\,j}{2} \le 0 \right] \end{cases}$$

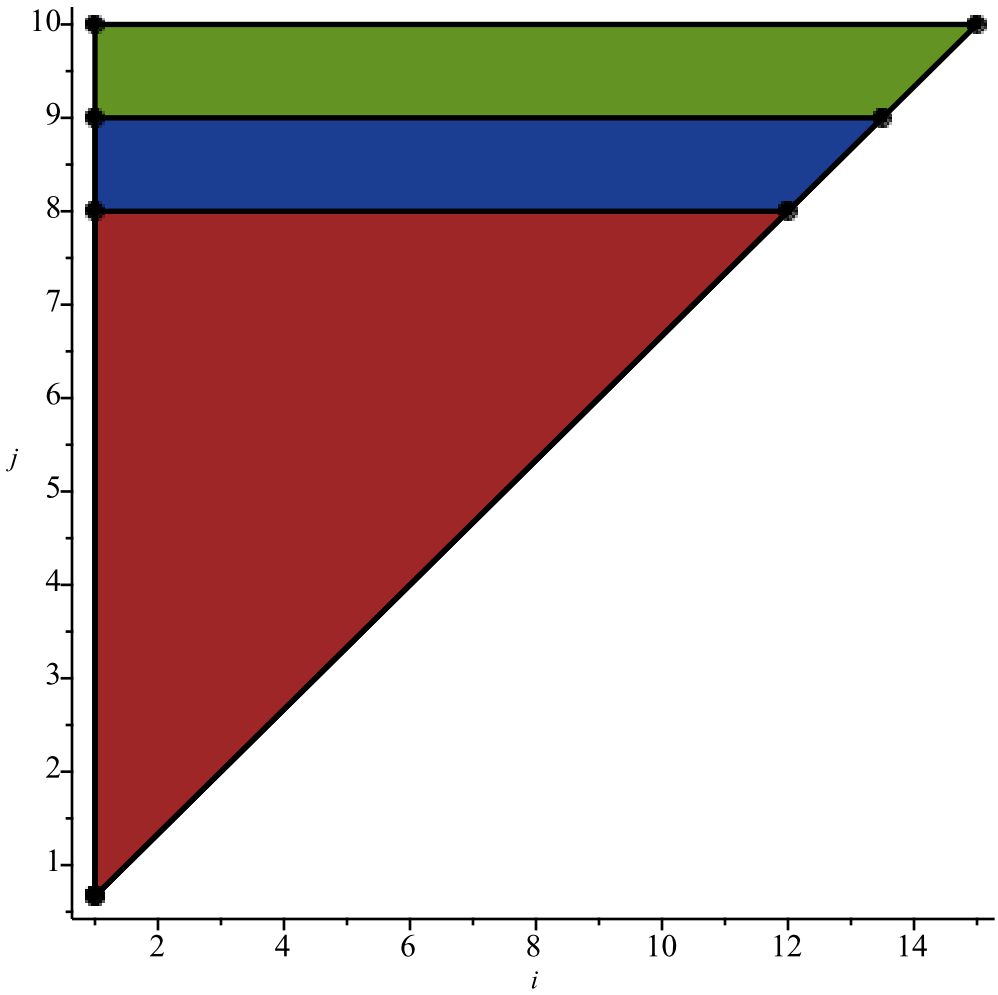
```
> ps_9 := PolyhedralSet(eval(parametric_triangle_2, n=9));
```

$$ps\_9 := \begin{cases} Coordinates & : & [i, j] \\ Relations & : & \left[j \le 9, -i \le -1, i - \frac{3\,j}{2} \le 0 \right] \end{cases}$$

```
> ps_10 := PolyhedralSet(eval(parametric_triangle_2, n=10));
```

$$ps\_10 := \begin{cases} Coordinates & : & [i, j] \\ Relations & : & \left[j \le 10, -i \le -1, i - \frac{3\,j}{2} \le 0 \right] \end{cases}$$

```
> Plot([ps_8, ps_9, ps_10]);
```




```
> NumberOfIntegerPoints(ps_8);
```

52

```
> NumberOfIntegerPoints(ps_9);
```

65

```
> NumberOfIntegerPoints(ps_10);
```

80

```
> NumberOfIntegerPoints(parametric_triangle_2, [i, j], [n], output =
  piecewise);
```

$$\begin{cases} \left\langle \text{one of the polynomials } \begin{pmatrix} \frac{3}{4}n^2+\frac{1}{2}n \\ -\frac{1}{4}+\frac{3}{4}n^2+\frac{1}{2}n \end{pmatrix} \text{ depending on the value of } n \text{ modulo } 2 \right\rangle & 0 \le -2+ \\ 0 & otherwis \end{cases}$$

- This computation gives rise to a so-called *quasi-polynomial*. A quasi-polynomial is an expression given by an integer m (the *modulo*), a list of m polynomials $[f_1,\ldots,f_m]$ (the *constituents*), and a so-called *switch* s, which is a linear polynomial. The expression takes on the value f_i whenever s is equal to i modulo m. In other words, it is a polynomial with coefficients that are periodic functions, all with the same integral period. In this case, $m=2$ and $s=n$: the result is given by one of two polynomials, depending on the parity of n.
- The easiest way to deal with such quasi-polynomials is when they occur inside ValuesUnderConstraints objects, described in a section below, which the NumberOfIntegerPoints command generates by default for parametric polyhedral sets.

```
> np := NumberOfIntegerPoints(parametric_triangle_2, [i, j], [n]);
```

$$np := \left[\left\langle \text{value} \left\{ \left\langle \text{one of the polynomials} \begin{pmatrix} \frac{3}{4}n^2+\frac{1}{2}n \\ -\frac{1}{4}+\frac{3}{4}n^2+\frac{1}{2}n \end{pmatrix} \text{depending on the value of } n \right. \right. \right.$$
$$\left. \left. \left. \text{modulo } 2 \right\rangle \right\} \text{when } [0 \le -2+3n] \right\rangle \left.\right]$$

```
> with(ValuesUnderConstraints):
> Eval(np[1], n=8);
```

$$\{52\}$$

```
> Eval(np[1], n=9);
```

$$\{65\}$$

```
> Eval(np[1], n=10);
```

$$\{80\}$$

- The polyhedral set below depends on two parameters, m and n. The shape can be a triangle or a quadrangle, depending on the parameter values. In the display below, all shapes extend to the lower left; the blue region contains the red, and the green contains both.

```
> parametric_polytope := [1 <= i, j <= n,  i <= m, 3*i <= 5*j];
```

$$parametric\_polytope := [1 \le i, j \le n, i \le m, 3\,i \le 5\,j]$$

```
> ps23 := PolyhedralSet(eval(parametric_polytope, {m=2, n=3}));
```

$$ps23 := \begin{cases} Coordinates & : & [i, j] \\ Relations & : & \left[j \le 3, -i \le -1, i - \frac{5\,j}{3} \le 0, i \le 2\right] \end{cases}$$

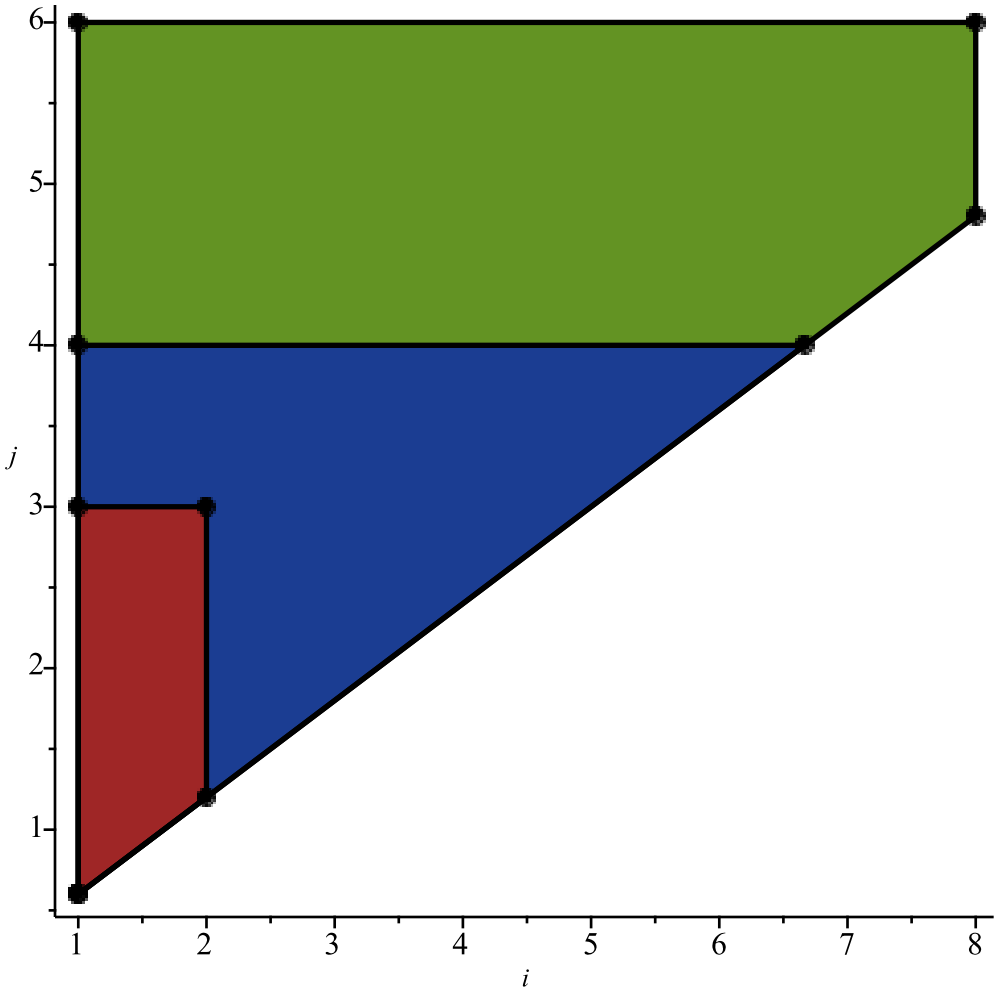
```
> ps74 := PolyhedralSet(eval(parametric_polytope, {m=7, n=4}));
```

$$ps74 := \begin{cases} Coordinates & : & [i, j] \\ Relations & : & \left[j \le 4, -i \le -1, i - \frac{5\,j}{3} \le 0\right] \end{cases}$$

```
> ps86 := PolyhedralSet(eval(parametric_polytope, {m=8, n=6}));
```

$$ps86 := \begin{cases} Coordinates & : & [i, j] \\ Relations & : & \left[j \le 6, -i \le -1, i - \frac{5\,j}{3} \le 0, i \le 8\right] \end{cases}$$

```
> Plot([ps23, ps74, ps86]);
```




```
> map(NumberOfIntegerPoints, [ps23, ps74, ps86]);
```

$$[5, 15, 31]$$

- We first view the number of integer points, which is easiest as follows. Then we evaluate the results for the configurations we displayed above.

```
> NumberOfIntegerPoints(parametric_polytope, [i, j], [m, n], output
  = piecewise);
```

$$\left\{\left\{n\,m+\left\langle \text{one of the polynomials}\left(\begin{array}{c}0\\0\\-\frac{2}{5}\\-\frac{1}{5}\\-\frac{2}{5}\end{array}\right)\text{depending on the value of } m \text{ modulo } 5\right\rangle-\frac{3\,m^2}{10}+\frac{3\,m}{10}\right\}\quad 0\right.$$

$$\{n\}$$

$$\left\{\left\{\left\langle \text{one of the polynomials}\left(\begin{array}{c}0\\-\frac{1}{3}\\-\frac{1}{3}\end{array}\right)\text{depending on the value of } n \text{ modulo } 3\right\rangle+\frac{5\,n^2}{6}+\frac{n}{2}\right\}\right.$$

$$\left.\left.\left\{\left\langle \text{one of the polynomials}\left(\begin{array}{c}0\\-\frac{1}{3}\\-\frac{1}{3}\end{array}\right)\text{depending on the value of } n \text{ modulo } 3\right\rangle+\frac{5\,n^2}{6}+\frac{n}{2}\right\}\right\}\right\}$$

```
> np := NumberOfIntegerPoints(parametric_polytope, [i, j], [m, n]);
```

$$np := \left[\left\langle \text{value}\left\{ n\,m + \left\langle \text{one of the polynomials} \begin{pmatrix} 0 \\ 0 \\ -\frac{2}{5} \\ -\frac{1}{5} \\ -\frac{2}{5} \end{pmatrix} \text{depending on the value of } m \text{ modulo } 5 \right\rangle \right.\right.$$

$$\left. - \frac{3\,m^2}{10} + \frac{3\,m}{10} \right\} \text{when } [0 \le m - 2, 0 \le 5\,n - 7, 0 \le -3\,m + 5\,n - 1] \Bigg\rangle, \langle \text{value } \{n\} \text{ when } [$$

$$-1 + m = 0, 0 \le 5\,n - 4]\rangle,$$

$$\left\langle \text{value}\left\{ \left\langle \text{one of the polynomials} \begin{pmatrix} 0 \\ -\frac{1}{3} \\ -\frac{1}{3} \end{pmatrix} \text{depending on the value of } n \text{ modulo } 3 \right\rangle + \frac{5\,n^2}{6} \right.\right.$$

$$\left\langle \left\{ + \frac{n}{2} \right\} \text{ when } [-3m + 5n = 0, 0 \le 5n - 4] \right\rangle,$$

$$\left\langle \text{value} \left\{ \left\langle \text{one of the polynomials} \begin{pmatrix} 0 \\ -\frac{1}{3} \\ -\frac{1}{3} \end{pmatrix} \text{ depending on the value of } n \text{ modulo } 3 \right\rangle + \frac{5n^2}{6} \right.$$

$$\left. \left\{ + \frac{n}{2} \right\} \text{ when } [0 \le 5n - 4, 0 \le 3m - 5n - 1] \right\rangle \right]$$

```
> map(Eval, np, {m=2, n=3});
```

$$[\{5\}]$$

```
> map(Eval, np, {m=7, n=4});
```

$$[\{15\}]$$

```
> map(Eval, np, {m=8, n=6});
```

$$[\{31\}]$$

- The following 3-dimensional polyhedral set depends on two parameters, m and n. For this example, the geometry varies with the values of the parameters.

```
> parametric_polytope_2 := [1 <= i, i <= n, j <= m, 1 <= j, j <= i,
  1 <= k, k <= j];
```

$$parametric\_polytope\_2 := [1 \le i, i \le n, j \le m, 1 \le j, j \le i, 1 \le k, k \le j]$$

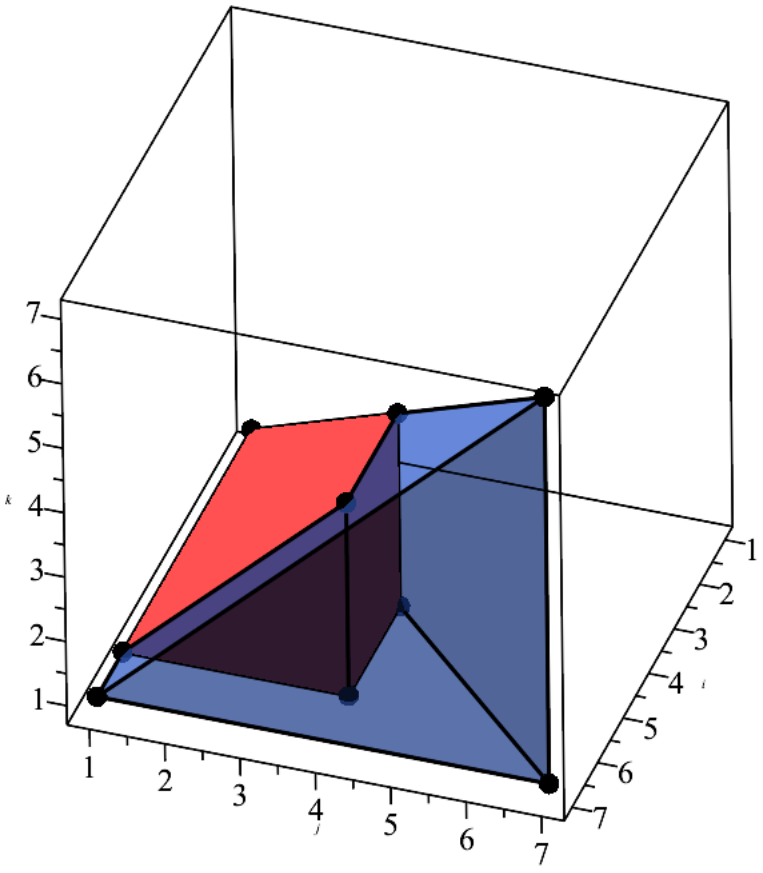
```
> ps46 := PolyhedralSet(eval(parametric_polytope_2, {m=4, n=6}));
```

$$ps46 := \begin{cases} Coordinates & : & [i, j, k] \\ Relations & : & [-k \le -1, -j + k \le 0, j \le 4, -i + j \le 0, i \le 6] \end{cases}$$

```
> ps97 := PolyhedralSet(eval(parametric_polytope_2, {m=9, n=7}));
```

$$ps97 := \begin{cases} Coordinates & : & [i, j, k] \\ Relations & : & [-k \le -1, -j + k \le 0, -i + j \le 0, i \le 7] \end{cases}$$

```
> Plot([ps46, ps97], transparency=[0, 0.5], orientation=[18, 54, -1]
  );
```




```
> NumberOfIntegerPoints(ps46);
```

40

```
> NumberOfIntegerPoints(ps97);
```

84

```
> np := NumberOfIntegerPoints(parametric_polytope_2, [i, j, k], [m,
  n], output = piecewise);
```

$$np := \begin{cases} \left\{\frac{1}{3}n + \frac{1}{2}n^2 + \frac{1}{6}n^3\right\} & -m+n=0 \wedge 0 \le -2+n \\ \{1\} & n-1=0 \wedge -m+1=0 \\ \left\{\frac{1}{3}n + \frac{1}{2}n^2 + \frac{1}{6}n^3\right\} & 0 \le -2+n \wedge 0 \le m-n-1 \\ \{1\} & n-1=0 \wedge 0 \le m-2 \\ \left\{\frac{1}{3}m - \frac{1}{3}m^3 + \frac{1}{2}n\,m^2 + \frac{1}{2}n\,m\right\} & 0 \le m-2 \wedge 0 \le -3+n \wedge 0 \le -m+n-1 \\ \{4+n\} & -m+1=0 \wedge 0 \le -2+n \end{cases}$$

- In this case, the result does not contain any quasi-polynomials.

```
> eval(np, {m=4, n=6});
```

$$\{40\}$$

```
> eval(np, {m=9, n=7});
```

$$\{84\}$$

- The examples in this section come from the following reference:

  Rui-Juan Jing, Yuzhuo Lei, Christopher F. S. Maligec, Marc Moreno Maza: **Counting the Integer Points of Parametric Polytopes: A Maple Implementation**. In the proceedings of *Computer Algebra in Scientific Computing: 26th International Workshop*, pp. 140-160.

VerifyTools

- In earlier versions of Maple, testing and verifying properties could be done using the verify command which came with a number of useful pre-built verification methods. For Maple 2025, we introduce the package VerifyTools which allows users to extend this system to include their own personally defined verifications. While any given verification could be simulated by writing an equivalent procedure, the great advantage of using VerifyTools is the ability to combine various user-defined and/or system verifications together in a structured verification.
- VerifyTools is similar to the TypeTools package. A type is essentially a predicate that a single expression can either satisfy or not. Analogously, a verification is a predicate that applies to a pair of expressions, comparing them. Just as types can be combined to produce compound types, verifications can also be combined to produce compound, or structured, verifications. New types can be created, retrieved, queried, or,deleted using the commands AddType, GetType (or GetTypes), Exists, and RemoveType, respectively.

Similarly in the VerifyTools package we can create, retrieve, query, or delete verifications using AddVerification, GetVerification (or GetVerifications), Exists, and RemoveVerification.

- The package command VerifyTools:-Verify is also available as the top-level Maple command verify which should already be familiar to expert Maple users. Similarly, the command VerifyTools:-IsVerification is also available as a type, that is,

```
> VerifyTools:-IsVerification(ver);
```

$$false$$

will return the same as

```
> type(ver, 'verification');
```

$$false$$

- In the following examples we show what can be done with these commands.

```
> with(VerifyTools):
```

- Suppose we want to create a verification which checks that the length of a result has not increased compared to the expected result. We can do this using the AddVerification command:

```
> AddVerification(length_not_increased, (a, b) -> evalb(length(a) <=
  length(b)));
```

- First, we can check the existence of our new verification and get its value:

```
> Exists(length_not_increased);
```

$$true$$

```
> GetVerification(length_not_increased);
```

$$(a, b) \mapsto evalb(length(a) \leq length(b))$$

- For named verifications, IsVerification is equivalent to Exists (since names are only recognized as verifications if an entry exists for them in the verification database):

```
> IsVerification(length_not_increased);
```

$$true$$

- On the other hand, a nontrivial structured verification can be checked with IsVerification,

```
> IsVerification(boolean = length_not_increased);
```

$$true$$

whereas Exists only accepts names:

```
> Exists(boolean = length_not_increased);
```

Error, invalid input: VerifyTools:-Exists expects its 1st argument, x, to be of type symbol, but received boolean = length not increased

• The preceding command using Exists is also equivalent to the following type call:

```
> type(boolean = length_not_increased, verification);
```

true

• Now, let's use the new verification:

```
> Verify(x + 1/x, (x^2 + 1)/x, length_not_increased);
```

true

```
> Verify((x^2 + 1)/x, x + 1/x, length_not_increased);
```

false

• For a more complicated example, suppose we have two lists of results, one expected and the other newly computed:

```
> expected_results := [(x^2+1)/x, 2*sin(t)*cos(t)];
```

$$expected\_results := \left[\frac{x^2+1}{x}, 2\sin(t)\cos(t)\right]$$

```
> computed_results := [sin(2*t), x + 1/x];
```

$$computed\_results := \left[\sin(2t), x + \frac{1}{x}\right]$$

We want to verify two things: that the computed results are mathematically equal to the expected results, though they may have a different form (we can use the simplify verification for this), and that the length of the computed results are not larger than those expected (we use our custom verification, length_not_increased, for this). Moreover, the entries in the lists can be in a different order, so we can use the as_set verification for this. Putting all of this together, we would use the following structured verification:

```
> verify(computed_results, expected_results, as_set(And(simplify,
  length_not_increased)));
```

true

In previous versions of Maple, we could have written a procedure to do this work, but we would have had to re-implement at least the as_set verifications: it works only when the relation between the elements of the "sets" can be expressed as a verification.

• Finally, let's remove the verification:

```
> RemoveVerification(length_not_increased);
```

```
> Exists(length_not_increased);
```

false

```
> GetVerification(length_not_increased);
```

Error, (in VerifyTools:-GetVerification) length not increased is not a recognized verification

- GetVerifications returns the list of all verifications known to the system:

```
> GetVerifications();
```

[*&under*, *Array*, *FAIL*, *FrobeniusGroupId*, *Global*, *Matrix*, *MultiSet*, *PermGroup*, *RootOf*, *SmallGroupId*, *Vector*, *address*, *after*, *approx*, *array*, *as_list*, *as_multiset*, *as_set*, *attributes*, *boolean*, *box*, *cbox*, *curve*, *curves*, *dataframe*, *dataseries*, *default*, *default*, *dummyvariable*, *equal*, *evala*, *evalc*, *expand*, *false*, *float*, *function*, *function_bounds*, *function_curve*, *function_shells*, *greater_equal*, *greater_than*, *in_convex_polygon*, *indef_int*, *interval*, *less_equal*, *less_than*, *list*, *listlist*, *matrix*, *member*, *multiset*, *neighborhood*, *neighbourhood*, *normal*, *permute_elements*, *plot*, *plot3d*, *plot_distance*, *plotthing_compile_result*, *polynom*, *procedure*, *ptbox*, *range*, *rational*, *record*, *relation*, *reverse*, *rifset*, *rifsimp*, *rtable*, *set*, *sign*, *simplify*, *sublist*, *subset*, *subtype*, *superlist*, *superset*, *supertype*, *symbol*, *table*, *table_indices*, *testeq*, *text*, *true*, *truefalse*, *type*, *undefined*, *units*, *vector*, *verifyfunc*, *wildcard*, *xmltree*, *xvm*]

Note: VerifyTools has been available in Maple for roughly 24 years, but until now it has never been documented, as it was originally intended for internal use only. Some of the VerifyTools commands have been revised for this release.

Improvements in Generating Random Expressions

- The RandomTools:-GenerateSimilar command can produce a new expression that shares characteristics with a given expression. One use of this is in generating similar problems for student practice. With Maple 2025, we've improved this capability to include making it even easier for students to practice differentiation problems by producing a new differentiation problem that requires the same core steps required to find the solution as the given problem.
- In addition, for convenience an option has been added to facilitate the creation of a list of random expressions instead of just one. For details, see Improved Ability to Generate Similar Problems.

Improved Units Support

- max and min now offer built-in Units support for simple inputs. Prior to Maple 2025 it

was required that one of the Units packages, for example, Units:-Simple, be loaded, otherwise max and min would return unevaluated

```
> max(3*Unit(m),2*Unit(m));
```

$$3\ \mathrm{m}$$

- The sort command now supports inputs containing Units.

```
> sort([3*Unit(m),4*Unit(ft),10*Unit(ft)]);
```

$$[4\ \mathrm{ft}, 3\ \mathrm{m}, 10\ \mathrm{ft}]$$

- The fdiff command now supports units as well.

```
> fdiff( 2.3*Unit(m/s)*t + 1/2 * 9.81 * Unit(m/s^2)*t^2, t=2.1*Unit
  (s) );
```

$$22.90100000\ \frac{\mathrm{m}}{\mathrm{s}}$$

List and Vector Form for fsolve

- The fsolve command now accepts its equations, variables, and ranges options in list or Vector form. The form in which the variables are specified determines the form for the returned solutions. Using list or Vector form to specify the variables will preserve their order in the solutions. If the variables are specified in Vector form then the solution contains the ordered numeric values rather than equations for each variable. For the following example the result is a list in which the y-value solution appears first, matching the order of the specified list of variable names.

```
> fsolve(<x^2-y-3=0, x+y=4>, [y,x], {y=4..8});
```

$$[y = 7.192582404, x = -3.192582404]$$

Ceil, Floor, Round, Trunc

- Four new top-level commands have been added that round to integer multiples of a given second argument. Each of these has the same behavior as the corresponding lowercase version of the command. More details and examples on the Trunc help page. Some common use cases of these commands would be round to multiples of 10, multiples of π, or to do conversions to nearest integer multiples of units.

```
> restart;
```

```
> Floor(72*Pi, 10);
```

$$220$$

```
> Floor(72*Pi, 0.01);
```

$$226.19$$

```
> Ceil(27, Pi);
```

$$9\,\pi$$

```
> Trunc( 14, sqrt(3) );
```

$$8\,\sqrt{3}$$

```
> Round( 45*Unit(miles), Unit(km) );
```

$$72\ \mathrm{km}$$

Identifying an Integer Sequence from Its First Terms

- The new command IdentifySequence attempts to find a formula for the nth term of the sequence of integers that you input.

```
> IdentifySequence([1, 3, 5, 7, 9], 'n');
```

$$2\,n-1$$

- For more information, see Find a Formula for the nth Term of a Given Integer Sequence.

Improvements to evala

- Two new primitives RealPart and ImaginaryPart have been added to the evala command. These compute a RootOf expression of smallest possible degree for the real or imaginary part(s), respectively, of a given RootOf.

```
> f := RootOf(x^4+x-1, 'index'=4):
```

```
> re, im := evala(RealPart(f)), evala(ImaginaryPart(f));
```

$$re, im := RootOf\left(64\,\_Z^6+16\,\_Z^2-1, index=real_2\right), RootOf\left(4096\,\_Z^{12}-2048\,\_Z^8-1664\,\_Z^6-1792\,\_Z^4+864\,\_Z^2-283, index=real_1\right)$$

```
> evalf(f), evalf(re), evalf(im);
```

$$0.2481260628-1.033982061\,I,\ 0.2481260628,\ -1.033982061$$

In earlier versions of Maple, the evalc@Re and evalc@Im calling sequences provide similar functionality. However, in this example they return radicals instead of RootOf

expressions.

```
> # evala(Re(f)), evala(Re(g));
```

$$\frac{\sqrt{6}\sqrt{\frac{\left(108+12\sqrt{849}\right)^{2/3}-48}{\left(108+12\sqrt{849}\right)^{1/3}}}}{12},$$

$$-\frac{1}{12}\left|\sqrt{6}\right.$$

$$\left(\left(\sqrt{\frac{\left(108+12\sqrt{849}\right)^{2/3}-48}{\left(108+12\sqrt{849}\right)^{1/3}}}\left(108+12\sqrt{849}\right)^{2/3}+12\sqrt{6}\left(108\right.\right.\right.$$

$$\left.\left.+12\sqrt{849}\right)^{1/3}-48\sqrt{\frac{\left(108+12\sqrt{849}\right)^{2/3}-48}{\left(108+12\sqrt{849}\right)^{1/3}}}\right)\Big/\left(\left(108+12\sqrt{849}\right)^{1/}\right.$$

$$\left.\left.{}^{3}\sqrt{\frac{\left(108+12\sqrt{849}\right)^{2/3}-48}{\left(108+12\sqrt{849}\right)^{1/3}}}\right)\right)^{1/2}\Bigg|$$

- The new commands also work for a RootOf with interval selector.

```
> f := RootOf(x^4+x-1, -2*I..1/2);
```

$$f := RootOf\left(\_Z^4+\_Z-1,\ -2\,\mathrm{I}..\frac{1}{2}\right)$$

```
> re, im := evala(RealPart(f)), evala(ImaginaryPart(f));
```

$$re, im := RootOf\left(64\,\_Z^6+16\,\_Z^2-1,\ 0..\frac{1}{2}\right), RootOf\left(4096\,\_Z^{12}-2048\,\_Z^8-1664\,\_Z^6\right.$$

$$\left.-1792\,\_Z^4+864\,\_Z^2-283,\ -2..0\right)$$

```
> evalf(f), evalf(re), evalf(im);
```

$$0.2481260628 - 1.033982061\,I,\ 0.2481260628,\ -1.033982061$$

- The new commands accept a RootOf without a selector. The defining polynomial of the resulting RootOf has all the real/imaginary parts of the input as roots, but in general has additional roots.

```
> g := RootOf(x^4+x-3);
```

$$g := RootOf\left(\_Z^4 + \_Z - 3\right)$$

```
> re, im := evala(RealPart(g)), evala(ImaginaryPart(g));
```

$$re,\ im := RootOf\left(64\,\_Z^{10} + 64\,\_Z^7 - 144\,\_Z^6 - \_Z^4 + 48\,\_Z^3 - 144\,\_Z^2 - \_Z + 3\right),$$
$$RootOf\left(4096\,\_Z^{13} - 6144\,\_Z^9 - 1664\,\_Z^7 - 16128\,\_Z^5 + 2592\,\_Z^3 - 6939\,\_Z\right)$$

```
> fsolve(op(g), 'complex');
```

$$-1.452626879,\ 0.1442958693 - 1.324149775\,I,\ 0.1442958693 + 1.324149775\,I,\ 1.164035140$$

The second real root of the following polynomial closest to the origin does not correspond to a real part.

```
> fsolve(op(re));
```

$$-1.452626879,\ -0.1442958693,\ 0.1442958693,\ 1.164035140$$

```
> fsolve(op(im));
```

$$-1.324149775,\ 0.,\ 1.324149775$$

```
> fsolve(op(im), 'complex');
```

$$-1.324149775,\ -0.6620748874 - 0.7984613741\,I,\ -0.6620748874 - 0.5098696355\,I,$$
$$-0.6620748874 + 0.5098696355\,I,\ -0.6620748874 + 0.7984613741\,I,\ -1.308331010\,I,\ 0.,$$
$$1.308331010\,I,\ 0.6620748874 - 0.7984613741\,I,\ 0.6620748874 - 0.5098696355\,I,\ 0.6620748874$$
$$+ 0.5098696355\,I,\ 0.6620748874 + 0.7984613741\,I,\ 1.324149775$$

In this example, evalc used to return RootOfs with higher degree.

```
> # r := [evalc(Re(g)),evalc(Im(g))];
```

$$r := \left[RootOf\left(4096\,\_Z^{16} + 4096\,\_Z^{13} - 6144\,\_Z^{12} - 128\,\_Z^{10} + 6144\,\_Z^9 - 16128\,\_Z^8 - 128\,\_Z^7\right.\right.$$
$$\left.+ 288\,\_Z^6 + 2304\,\_Z^5 - 6911\,\_Z^4 - 96\,\_Z^3 + 288\,\_Z^2 + \_Z - 3\right), RootOf\left(4096\,\_Z^{16}$$
$$\left.\left. - 6144\,\_Z^{12} - 1664\,\_Z^{10} - 16128\,\_Z^8 + 2592\,\_Z^6 - 6939\,\_Z^4\right)\right]$$

```
> map(factor@op, r);
```

$$\left[\left(\_Z^4 + \_Z - 3\right)\left(64\,\_Z^6 + 48\,\_Z^2 - 1\right)^2,\ \_Z^4\left(4096\,\_Z^{12} - 6144\,\_Z^8 - 1664\,\_Z^6 - 16128\,\_Z^4\right.\right.$$

$$+ 2592\ \_Z^2 - 6939\Big)\Big]$$

- The Chebyshev polynomials of the first kind have the real parts of the roots of unity as roots.

```
> evala(RealPart(RootOf(x^(2*5)+1)));
```

$$RootOf\left(16\ \_Z^5 - 20\ \_Z^3 + 5\ \_Z\right)$$

```
> expand(ChebyshevT(5,_Z));
```

$$16\ \_Z^5 - 20\ \_Z^3 + 5\ \_Z$$

Here, the degree of the RootOf returned by evalc is much higher, and almost all of the factors are spurious.

```
> evalc(Re(RootOf(x^(2*5)+1)));
```

$$RootOf\Big(120892581961462917470617 6\ \_Z^{100} + 23611832414348226068480\ \_Z^{90} - 10648267724398377985638 40\ \_Z^{80} + 20471747113760048807936 0\ \_Z^{70} + 89786788197682150113280\ \_Z^{60} + 549575213307508922777 6\ \_Z^{50} - 6062575320956928000 0\ \_Z^{40} + 153936696153600000\ \_Z^{30} + 2456009765625\ \_Z^{20} + 9765625\ \_Z^{10}\Big)$$

```
> factor(op(Function Call));
```

$$\_Z^{10}\left(\_Z^2+1\right)\left(\_Z^8-\_Z^6+\_Z^4-\_Z^2+1\right)\left(16\ \_Z^4-20\ \_Z^2+5\right)^2\left(256\ \_Z^8+160\ \_Z^4+100\ \_Z^2+25\right)^2\left(256\ \_Z^8+320\ \_Z^6+160\ \_Z^4+25\right)^2\left(16\ \_Z^4+12\ \_Z^2+1\right)^2\left(256\ \_Z^8-256\ \_Z^6+96\ \_Z^4+4\ \_Z^2+1\right)^2\left(256\ \_Z^8+64\ \_Z^6+96\ \_Z^4-16\ \_Z^2+1\right)^2$$

ExpressionTools

- A new package ExpressionTools was created which provides some tools for comparing expressions and highlighting their differences. For more information, see Tools for Comparing Expressions

ValuesUnderConstraints

- The ValuesUnderConstraints package is a new Maple package for Maple 2025, which is designed to support the implementation of algorithms with output, or values, depending on parameters. In other words, this new package facilitates the implementation of mechanisms for case discussion. The core concept in the package is

that of a pair consisting of one or more values (arbitrary algebraic expressions) together with constraints under which these values are valid. In those pairs, variables can be real-valued or integer-valued, while constraints can be equalities or inequalities between polynomials in those variables. While polynomials of arbitrary degree are supported, the current version of this package is optimized for linear constraints.

```
> with(ValuesUnderConstraints):
```

Comparing the domains of several functions

- Consider two functions $f(a, b, c, d)$ and $g(a, b, c, d)$, together with their domains of definition. When are both defined, and when is only one of them defined? The command MakeCaseDiscussion answers these questions by computing a partition of the union of the domains so that on each part of this partition either $f(a, b, c, d)$, or $g(a, b, c, d)$, or both are defined.

```
> vc1 := ValueUnderConstraints([f(a, b, c, d)], [a <> 0, b < 0, c >
  0, d <> 0]);
```

$$vc1 := \left\langle \text{value} \left\{ RootOf\left(\_Z^4 + \_Z - 1, -2\,\mathrm{I}..\frac{1}{2} \right) \right\} \text{when} \left[a \neq 0, d \neq 0, 0 < c, 0 < -b \right] \right\rangle$$

```
> vc2 := ValueUnderConstraints([g(a, b, c, d)], [c <> 0, d > 0, a <
  0, b <> 0]);
```

$$vc2 := \left\langle \text{value} \left\{ RootOf\left(\_Z^4 + \_Z - 3 \right) \right\} \text{when} \left[b \neq 0, c \neq 0, 0 < d, 0 < -a \right] \right\rangle$$

```
> MakeCaseDiscussion([vc1, vc2], output = piecewise);
```

$$\begin{cases} \left\{ RootOf\left(\_Z^4 + \_Z - 1, -2\,\mathrm{I}..\frac{1}{2} \right) \right\} & a \neq 0 \wedge 0 < c \wedge 0 < -b \wedge 0 < -d \\ \left\{ RootOf\left(\_Z^4 + \_Z - 1, -2\,\mathrm{I}..\frac{1}{2} \right) \right\} & 0 < a \wedge 0 < c \wedge 0 < d \wedge 0 < -b \\ \left\{ RootOf\left(\_Z^4 + \_Z - 3 \right), RootOf\left(\_Z^4 + \_Z - 1, -2\,\mathrm{I}..\frac{1}{2} \right) \right\} & 0 < c \wedge 0 < d \wedge 0 < -a \wedge 0 < -b \\ \left\{ RootOf\left(\_Z^4 + \_Z - 3 \right) \right\} & b \neq 0 \wedge 0 < d \wedge 0 < -a \wedge 0 < -c \\ \left\{ RootOf\left(\_Z^4 + \_Z - 3 \right) \right\} & 0 < b \wedge 0 < c \wedge 0 < d \wedge 0 < -a \end{cases}$$

• Consider four pairs of values and constraints.

```
> vc1 := ValueUnderConstraints([1], [a = 0, b >= 0, c < 0, d <> 0])
  ;
```

$$vc1 := \langle \text{value } \{1\} \text{ when } [a = 0, d \neq 0, 0 < -c, 0 \leq b] \rangle$$

```
> vc2 := ValueUnderConstraints([2], [b = 0, a >= 0, d <> 0, c > -1])
  ;
```

$$vc2 := \langle \text{value } \{2\} \text{ when } [b = 0, d \neq 0, 0 < c + 1, 0 \leq a] \rangle$$

```
> vc3 := ValueUnderConstraints([3], [a <> 0, b >= 0, c < 0, d = 0])
  ;
```

$$vc3 := \langle \text{value } \{3\} \text{ when } [d = 0, a \neq 0, 0 < -c, 0 \leq b] \rangle$$

```
> vc4 := ValueUnderConstraints([4], [b <> 0, a >= 0, d <> 0, c = 0])
  ;
```

$$vc4 := \langle \text{value } \{4\} \text{ when } [c = 0, b \neq 0, d \neq 0, 0 \leq a] \rangle$$

• Use MakeCaseDiscussion to check which of these four pairs have intersecting domains.

```
> MakeCaseDiscussion([vc1, vc2, vc3, vc4], output = piecewise);
```

$$\begin{cases} \{1\} & a = 0 \wedge d \neq 0 \wedge 0 < -c \wedge 0 \leq b \wedge 0 \leq -c - 1 \\ \{1\} & a = 0 \wedge d \neq 0 \wedge 0 < b \wedge 0 < -c \wedge 0 < c + 1 \\ \{1, 2\} & a = 0 \wedge b = 0 \wedge d \neq 0 \wedge 0 < -c \wedge 0 < c + 1 \\ \{2\} & b = 0 \wedge d \neq 0 \wedge 0 < c + 1 \wedge 0 \leq a \wedge 0 \leq c \\ \{2\} & b = 0 \wedge d \neq 0 \wedge 0 < a \wedge 0 < -c \wedge 0 < c + 1 \\ \{4\} & c = 0 \wedge b \neq 0 \wedge d \neq 0 \wedge 0 \leq a \\ \{3\} & d = 0 \wedge a \neq 0 \wedge 0 < -c \wedge 0 \leq b \end{cases}$$

Distinguishing between real-valued and integer-valued variables

• Let's modify the four pairs above and specify that their variables are integer-valued.

```
> vc1i := ValueUnderConstraints([1], [a = 0, b >= 0, c < 0, d <> 0],
  {a, b, c, d});
```

$$vc1i := \langle \text{value } \{1\} \text{ when } [a = 0, d \neq 0, 0 \leq b, 0 \leq -c - 1] \rangle$$

```
> vc2i := ValueUnderConstraints([2], [b = 0, a >= 0, d <> 0, c >
  -1], {a, b, c, d});
```

$$vc2i := \langle \text{value } \{2\} \text{ when } [b = 0, d \neq 0, 0 \leq a, 0 \leq c] \rangle$$

```
> vc3i := ValueUnderConstraints([3], [a <> 0, b >= 0, c < 0, d = 0],
  {a, b, c, d});
```

$$vc3i := \langle \text{value } \{3\} \text{ when } [d = 0, a \neq 0, 0 \leq b, 0 \leq -c - 1] \rangle$$

```
> vc4i := ValueUnderConstraints([4], [b <> 0, a >= 0, d <> 0, c =
  0], {a, b, c, d});
```

$$vc4i := \langle \text{value } \{4\} \text{ when } [c = 0, b \neq 0, d \neq 0, 0 \leq a] \rangle$$

- Use MakeCaseDiscussion and observe the difference between the result here and the result above for the real-valued case.

```
> MakeCaseDiscussion([vc1i, vc2i, vc3i, vc4i], output = piecewise);
```

$$\begin{cases} \{4\} & c = 0 \wedge b \neq 0 \wedge d \neq 0 \wedge 0 \leq a \\ \{2\} & b = 0 \wedge d \neq 0 \wedge 0 \leq a \wedge 0 \leq c \\ \{1\} & a = 0 \wedge d \neq 0 \wedge 0 \leq b \wedge 0 \leq -c - 1 \\ \{3\} & d = 0 \wedge a \neq 0 \wedge 0 \leq b \wedge 0 \leq -c - 1 \end{cases}$$