Performance Improvements in Maple 2025

Compact Hardware Float Representation

- Maple 2025 uses a new compact representation of double-precision hardware floats. The new data structure works on 64-bit machines by fitting the double into an address word when possible. This is referred to as an "immediate"-double, similar to the concept of "immediate"-integers that has been in place in Maple for many versions. The result is that extracting an entry from a **datatype=float[8]** Array does not need to allocate memory. This can lead to a significant memory saving and performance improvements.
- The following example used about 1.2s of CPU time and 159MB of memory on a benchmark machine in the previous version of Maple. It now takes 0.16s of CPU time and uses only 11MB of memory. This is an order of magnitude better on both the time and space measures.

```
> n := 100:
```

```
> A := Array(1..64-1, 1..n, datatype=float[8]):
```

```
> tt, tr, bu := time(), time[real](), kernelopts(bytesused):
```

```
> for xi from 2 to 64 - 1 do
     for xj from 2 to n do
         A[xi, xj] := add('floor(log[2](1+A[xi-1,xj-1]))', i=1..xi);
     end do;
  end do:
```

```
> time() - tt, time[real]() - tr, kernelopts(bytesused)-bu;
```

$$0.275,\ 0.276,\ 12027664 \qquad (1)$$

PopCount

- Sometimes you want to know how many bits in the binary representation of a nonnegative integer are 1. This is often called the **PopCount** of the integer.
- Maple 2025 now supports this operation with the Bits:-PopCount command.
- This number has a binary representation that consists of 100000 repetitions of the pattern 1, 1, 0. So it contains 200000 bits that are 1.

```
> n := 6 * (8^100000 - 1) / (8 - 1):
```

```
> Bits:-PopCount(n);
```

$$200000 \quad (2)$$

- Doing this in earlier versions of Maple would have required something like the following. The new **PopCount** command is about 5000x faster.

```
> add(Bits:-Split(n));
```

$$200000 \quad (3)$$

Numeric Evaluation of Special Functions with Constant Arguments

- By default Maple returns a symbolic answer for exact quantities like sin(3) and sqrt(2). A new setting can be used to cause evaluation of some built-in special functions when given integer constant arguments to be evaluated to hardware floating-point results, provided Digits and UseHardwareFloats are set appropriately. The functions that use this setting are a subset of the list given in evalhf/fcnlist. Using this can improve performance when the final result is expected to be numeric.

```
> sin(3), sqrt(2), log(2);
```

$$\sin(3), \sqrt{2}, \ln(2) \quad (4)$$

```
> kernelopts(evalhfconstspecfun=true):
```

```
> sin(3), sqrt(2), log(2);
```

$$0.141120008059867, 1.41421356237310, 0.693147180559945 \quad (5)$$

Array Output for Numeric Initial Value Problems

- The default output type for numeric IVP and DAE solution (procedurelist) has been extended for the core solvers (rkf45, ck45 and rosenbrock) to support Array-form output.

This allows for highly efficient solution of parameterized problems, especially when the compile option is in use, as all data required for a parameter set can be obtained in a single call.

```
> dsn := dsolve({diff(y(t),t,t)+a*y(t)=0, y(0)=0, D(y)(0)=d},
  numeric, parameters=[d,a], compile=true);
```

$$dsn := \mathbf{proc}(x\_rkf45) \ \ldots \ \mathbf{end\ proc} \quad (6)$$

```
> dsn(parameters=[2,1]);
```

(7)

$$[d = 2., a = 1.] \quad (7)$$

List of the columns in the Array output (indexed from 0 for time)

```
> dsn("Array");
```

$$\left[t, y(t), \frac{\mathrm{d}}{\mathrm{d}t}\, y(t), d, a\right] \quad (8)$$

```
> res21 := dsn("Array"=[10,101]);
```

res21 := (9)

0.	0.	2.	2. ⋯
0.100000000000000	0.199666833375926	1.99000833251023	2. ⋯
0.200000000000000	0.397338664676613	1.96013316208172	2. ⋯
0.300000000000000	0.591040439869095	1.91067298936890	2. ⋯
0.400000000000000	0.778836734664567	1.84212199610204	2. ⋯
0.500000000000000	0.958851090966070	1.75516515691155	2. ⋯
0.600000000000000	1.12928502201017	1.65067126231218	2. ⋯
0.700000000000000	1.28843547071908	1.52968439610006	2. ⋯
0.800000000000000	1.43471226444675	1.39341349443144	2. ⋯
0.900000000000000	1.56665401260837	1.24321993845520	2. ⋯

1 .. 101 × 0 .. 4 Array

```
> dsn(parameters=[1,1]);
```

$$[d = 1., a = 1.] \quad (10)$$

```
> res11 := dsn("Array"=[10,101]);
```

res11 := (11)

0.	0.	1.	⋯
0.100000000000000	0.0998334180071792	0.995004167638281	⋯
0.200000000000000	0.198669339807089	0.980066582848396	⋯
0.300000000000000	0.295520228882018	0.955336493946141	⋯
0.400000000000000	0.389418348532907	0.921061006474038	⋯
0.500000000000000	0.479425562021517	0.877582581052255	⋯
0.600000000000000	0.564642524911161	0.825335623934648	⋯
0.700000000000000	0.644217712370945	0.764842224114780	⋯
0.800000000000000	0.717356174396688	0.696706724523328	⋯
0.900000000000000	0.783326966272817	0.621610011443851	⋯

1 .. 101 × 0 .. 4 Array

New implementation of the projection algorithm in PolyhedralSets

- A key part of the process of solving a system of linear inequalities with rational coefficients is the so-called Fourier-Motzkin Elimination algorithm. This is implemented in Maple in the PolyhedralSets package's Project command. In Maple 2025, we have added a new implementation of this algorithm in C++. In most cases, this algorithm is much faster than the previous implementation - up to several hundred times faster. Consider these two examples:

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

```
> ps1 := PolyhedralSet([
  -84*x0 - 108*x1 + 67*x2 - 43*x3 + 106*x4 - 14*x5 - 113*x6 - 31*x7
  <= 87,
  -89*x0 - 22*x1 - 24*x2 + 102*x3 - 24*x4 + 123*x5 - 17*x6 - 85*x7
  <= -88,
  70*x0 + 81*x1 + 52*x2 + 150*x3 - 36*x4 + 97*x5 + 109*x6 - 44*x7 <=
  36,
  51*x0 - 114*x1 + 12*x2 + 28*x3 + 35*x4 - 29*x5 - 33*x6 + 111*x7 <=
  -36,
  143*x0 - 1*x1 - 68*x2 + 109*x3 - 129*x4 - 147*x5 - 34*x6 + 47*x7
  <= 71,
```

```
-147*x0 - 137*x1 - 134*x2 - 7*x3 - 67*x4 - 6*x5 + 64*x6 + 53*x7 <=
103,
-126*x0 + 98*x1 + 47*x2 - 72*x3 - 147*x4 + 110*x5 + 112*x6 - 73*x7
<= -24,
-137*x0 + 93*x1 - 57*x2 - 28*x3 + 51*x4 + 43*x5 + 99*x6 + 16*x7 <=
42,
80*x0 + 128*x1 + 23*x2 - 117*x3 - 35*x4 + 9*x5 - 41*x6 + 89*x7 <=
97,
-54*x0 - 135*x1 - 77*x2 - 53*x3 - 146*x4 - 8*x5 + 77*x6 - 4*x7 <=
-106
]);
```

$$ps1 := \left\{ \begin{array}{lcl} Coordinates & : & [x0, x1, x2, x3, x4, x5, x6, x7] \\ Relations & : & \left[-x0 - \frac{5\,x1}{2} - \frac{77\,x2}{54} - \frac{53\,x3}{54} - \frac{73\,x4}{27} - \frac{4\,x5}{27} + \frac{77\,x6}{54} - \frac{2\,x7}{27} \le -\frac{53}{27}, \right. \end{array} \right.$$

```
> ps2 := PolyhedralSet([
107*x0 - 97*x1 + 41*x2 - 139*x3 + 124*x4 + 129*x5 + 87*x6 + 92*x7
+ 118*x8 - 57*x9 <= 108,
-27*x0 + 119*x1 + 88*x2 - 8*x3 - 99*x4 - 81*x5 + 48*x6 + 55*x7 +
57*x8 - 79*x9 <= -39,
36*x0 + 104*x1 + 15*x2 + 106*x3 - 88*x4 + 39*x5 + 138*x6 - 120*x7
+ 54*x8 - 75*x9 <= -2,
-2*x0 + 57*x1 - 16*x2 + 138*x3 + 86*x4 + 102*x5 + 55*x6 + 96*x7 -
114*x8 - 88*x9 <= 37,
-55*x0 - 74*x1 - 52*x2 - 15*x3 + 8*x4 - 8*x5 + 148*x6 + 96*x7 -
84*x8 + 44*x9 <= 62,
124*x0 + 36*x1 - 7*x2 - 69*x3 + 77*x4 + 33*x5 - 141*x6 - 114*x7 +
125*x8 + 27*x9 <= 24,
8*x0 + 64*x1 + 67*x2 - 25*x3 - 13*x4 - 22*x5 + 108*x6 - 108*x7 -
138*x8 + 13*x9 <= 108,
137*x0 + 86*x1 - 30*x2 - 102*x3 - 98*x4 + 17*x5 - 135*x6 - 12*x7 +
8*x8 - 95*x9 <= -77,
-126*x0 - 51*x1 - 79*x2 - 112*x3 + 119*x4 - 148*x5 + 141*x6 + 106*
x7 + 108*x8 - 105*x9 <= -114,
-112*x0 - 85*x1 + 8*x2 + 42*x3 - 134*x4 - 111*x5 + 17*x6 + 21*x7 -
45*x8 + 87*x9 <= 92,
53*x0 - 17*x1 - 78*x2 - 20*x3 - 76*x4 - 110*x5 + 144*x6 - 62*x7 +
54*x8 - 82*x9 <= -38,
```

```
  129*x0 - 71*x1 + 145*x2 + 96*x3 - 104*x4 - 123*x5 + 17*x6 - 67*x7
  - 137*x8 + 90*x9 <= -88,
  -116*x0 - 4*x1 - 14*x2 - 136*x3 - 2*x4 + 71*x5 + 82*x6 - 149*x7 -
  49*x8 + 24*x9 <= -37,
  135*x0 - 55*x1 + 130*x2 + 5*x3 - 73*x4 + 76*x5 + 74*x6 + 12*x7 -
  82*x8 + 61*x9 <= -77,
  -57*x0 - 20*x1 - 89*x2 + 15*x3 + 37*x4 + 93*x5 - 31*x6 - 32*x7 -
  77*x8 + 123*x9 <= 69
  ]);
```

$$ps2 := \begin{cases} Coordinates & : & [x0, x1, x2, x3, x4, x5, x6, x7, x8, x9] \\ Relations & : & \left[-x0 - \frac{85\,x1}{112} + \frac{x2}{14} + \frac{3\,x3}{8} - \frac{67\,x4}{56} - \frac{111\,x5}{112} + \frac{17\,x6}{112} + \frac{3\,x7}{16} - \frac{45\,x8}{112} + \right. \end{cases}$$

```
> CodeTools:-Usage(Project(ps1, [x6]));
```

```
memory used=13.14MiB, alloc change=24.00MiB, cpu time=337.00ms,
real time=234.00ms, gc time=159.59ms
```

$$\begin{cases} Coordinates & : & [x0, x1, x2, x3, x4, x5, x6, x7] \\ Relations & : & [x7 = 0, x5 = 0, x4 = 0, x3 = 0, x2 = 0, x1 = 0, x0 = 0] \end{cases} \quad \textbf{(14)}$$

```
> CodeTools:-Usage(Project(ps2, [x6]));
```

```
memory used=25.22MiB, alloc change=0 bytes, cpu time=536.00ms,
real time=536.00ms, gc time=0ns
```

$$\begin{cases} Coordinates & : & [x0, x1, x2, x3, x4, x5, x6, x7, x8, x9] \\ Relations & : & \left[x9 = 0, x8 = 0, x7 = 0, -x6 \le \frac{49999796901266130170006}{9803005266091256090959}, x6 \le -\frac{11866213904285374}{2426141831384629} \right. \end{cases}$$

- The calls above project the 8-dimensional polyhedral set $ps1$ and the 10-dimensional polyhedral set $ps2$ onto a one-dimensional space. On one machine, this took 24.8 and 217 seconds, respectively, in Maple 2024, but only 204 and 526 milliseconds in Maple 2025.

New implementation of transversals in Hypergraphs

- In Maple 2024, we introduced the Hypergraphs package, which deals with hypergraphs: a generalization of graphs where the "edges" can contain an arbitrary number of vertices. One of its features is computing is the transversal of a hypergraph: this is another hypergraph. This code was sped up significantly in 2025.
- To demonstrate, let us consider the so-called Lovasz hypergraphs. They are interesting

in part because they are isomorphic to their own transversal.

```
> with(Hypergraphs):
```

```
> with(ExampleHypergraphs):
```

```
> l5 := Lovasz(5);
```

$$l5 := \text{< a hypergraph on 15 vertices with 206 hyperedges >} \quad \textbf{(16)}$$

```
> l6 := Lovasz(6);
```

$$l6 := \text{< a hypergraph on 21 vertices with 1237 hyperedges >} \quad \textbf{(17)}$$

```
> l7 := Lovasz(7);
```

$$l7 := \text{< a hypergraph on 28 vertices with 8660 hyperedges >} \quad \textbf{(18)}$$

```
> CodeTools:-Usage(Transversal(l5));
```

```
memory used=188.43KiB, alloc change=0 bytes, cpu time=8.00ms,
real time=8.00ms, gc time=0ns
```

$$\text{< a hypergraph on 15 vertices with 206 hyperedges >} \quad \textbf{(19)}$$

```
> CodeTools:-Usage(Transversal(l6));
```

```
memory used=0.93MiB, alloc change=0 bytes, cpu time=494.00ms,
real time=493.00ms, gc time=0ns
```

$$\text{< a hypergraph on 21 vertices with 1237 hyperedges >} \quad \textbf{(20)}$$

```
> CodeTools:-Usage(Transversal(l7));
```

```
memory used=6.49MiB, alloc change=0 bytes, cpu time=2.39m, real
time=2.39m, gc time=0ns
```

$$\text{< a hypergraph on 28 vertices with 8660 hyperedges >} \quad \textbf{(21)}$$

- On one machine, computing the transversal of $l5$ took 1.19 seconds in Maple 2024 and 76 milliseconds in Maple 2025, a speedup of a factor 15. Computing the transversal of $l6$ took 2 minutes and 12 seconds in Maple 2024 and 447 milliseconds in Maple 2025, a speedup of roughly a factor 300. Computing the transversal of $l7$ in Maple 2025 takes 2 minutes and 8 seconds on this machine; computing it in Maple 2024, if it is successful at all, would likely take many days.
- This functionality is used in the Matroids package, which was also introduced in Maple 2024.

```
> with(Matroids):
```

```
> with(ExampleMatroids):
```

```
> c4 := NCubeMatroid(4);
```

$$c4 := \left\langle \text{the linear matroid whose ground set is the set of column vectors of the matrix: } \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \dots \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots \end{bmatrix} \text{ 5 × 16 Matrix} \right\rangle \quad \textbf{(22)}$$

```
> CodeTools:-Usage(numelems(Circuits(c4)));
```

```
memory used=41.75MiB, alloc change=0 bytes, cpu time=1.24s, real
time=1.14s, gc time=163.36ms
```

$$1348 \quad \textbf{(23)}$$

```
> CodeTools:-Usage(numelems(Hyperplanes(c4)));
```

```
memory used=451.04KiB, alloc change=0 bytes, cpu time=26.00ms,
real time=27.00ms, gc time=0ns
```

$$140 \quad \textbf{(24)}$$

- On one machine, computing the circuits of the four-dimensional cube matroid took 6.97 seconds in Maple 2024 and 953 milliseconds in Maple 2025. Computing the hyperplanes (after having computed the circuits!) took 12.8 seconds in Maple 2024 and 36 milliseconds in Maple 2025.