Performance Improvements in Maple 2023

Rational Linear Solver Speedup

- The underlying engine that solve uses to solve systems of linear equations with rational coefficients has been improved for many types of systems. The recently added SolveTools:-LinearSolvers:-RationalDense is now used for most systems of equations with more than 5 percent of their coefficients nonzero that are also overdetermined or highly underdetermined with user specified free variables.
- Here is an example of a highly overdetermined system with many solutions where the new method performs much better.

```
> N := 200: n := 100: r := rand(-1000..1000):
```

```
> M := Matrix(N+n, N, (i,j)->ifelse(j<(i-n), 0, r())):
```

```
> V1 := M . Vector([seq(r(),i=1..N)]):V2 := M . Vector([seq(r(),i=1..N)]):
```

```
> vars := [seq(cat(`x__`,i),i=1..N+1)]:
```

```
> sys := (lhs-rhs)~(LinearAlgebra:-GenerateEquations( <V1 | V2 | M>, vars)):
```

- The old default method

```
> CodeTools:-Usage(SolveTools:-LinearSolvers:-Rational(sys, indets(sys),dense=false) ):
```

```
memory used=67.95MiB, alloc change=8.80MiB, cpu time=760.00ms, real time=762.00ms, gc time=76.07ms
```

- The new default is significantly faster on this system

```
> CodeTools:-Usage(SolveTools:-LinearSolvers:-Rational(sys, indets(sys)) ):
```

```
memory used=27.13MiB, alloc change=-0.72MiB, cpu time=268.00ms, real time=269.00ms, gc time=43.79ms
```

Hardware Floating-Point Evaluation of Numeric Matrices

- When initializing hardware-float matrix, vector, and array data structures the kernel will opt for computing with numeric implementations of the initializer when possible. This results in dramatic speed-ups in some cases. The following example used to take several seconds and now returns instantly:

```
> tt:=time[real]():
> signal:=Vector(10^5,i->sin(i),datatype=float[8]):
> time[real]()-tt;
```

0.070

- A simple initialization of a float[8] rtable with integer values is now about 3.5 times faster.

```
> tt:=time[real]():
> v:=Vector(10^6, i->i, datatype=float[8]):
> time[real]()-tt;
```

0.085

- Evaluation of sums, products and functions now use less memory and a more direct path to hardware-float evaluation. The following example is more than two times faster in Maple 2023.

```
> N := 10^5:
> with(LinearAlgebra):
> X := RandomVector(N, generator = 0 .. 1.0, datatype = float[8]):
> Y := RandomVector(N, generator = 0 .. 1.0, datatype = float[8]):
> st := time[real]():
> total := Vector(N, i -> ifelse(X[i]^2 + Y[i]^2 < 1, 1, 0), datatype=
  float[8]):
> time[real]()-st;
```

0.501

- Type checking of datatype=float[8] rtables has also improved. The following example is hundreds of times faster in Maple 2023.

```
> A := [seq(Array(1..1,[i],datatype=float[8]),i=1..10^4)]:
> tt := time[real]():
> map(type,A,'rtable'(datatype=float[8])):
```

```
> time[real]() - tt;
```

0.010

- When a hardware-based rtable's elements are extracted they remain as a double-precision HFloat data-structure. Support for direct operation on these structures has been extended to include integer-output commands such as floor, round, and ceil. This gives a small improvement that adds up over millions of computations. Timings in the following example are now almost half of what they were in the previous version.

```
> n := 10^6:
```

```
> v :=  LinearAlgebra:-RandomVector(n,datatype=float[8]):
  tt := time():
  for i from 1 to numelems(v) do
     v[i] := floor(v[i]/10);
  end do:
```

```
> time()  - tt;
```

2.760

The Units Package

- The Units package, and in particular its Units:-Simple subpackage have received several upgrades that make them faster, sometimes several orders of magnitude faster.
- The Units:-Simple package works by calling the command Units:-TestDimensions every time an arithmetic operation is performed. This command has been re-implemented to be much faster. In user-submitted worksheets that heavily use units, we often see speedups between 3 times and 100 times for the whole worksheet.
- The following example shows a simple forward Euler discretization of constant acceleration for 2 seconds in 2000 steps.

```
> make_sys := proc(x :: symbol, v :: symbol, dt :: symbol, a ::
  symbol, n :: nonnegint, dtval :: algebraic, aval :: algebraic, $)
  local i;
      return [x[0] = 0, v[0] = 0, a = aval, dt = dtval,
          seq(v[i+1] = v[i] + dt * a, i = 0 .. n-1),
          seq(x[i+1] = x[i] + dt * v[i], i = 0 .. n-1)];
  end proc;
```

$$make\_sys := \mathbf{proc}(x::symbol, v::symbol, dt::symbol, a::symbol, n::nonnegint, dtval::algebraic, aval::algebraic, \$)$$

$$\mathbf{local}\ i;$$

$$\mathbf{return}\ [x[0] = 0, v[0] = 0, a = aval, dt = dtval, seq(v[i+1] = v[i] + dt*a, i = 0..n-1), seq(x[i+1] = x[i] + dt*v[i], i = 0..n-1)]$$

$$\mathbf{end\ proc}$$

```
> sys := make_sys(x, v, dt, a, 2000, 0.001*Unit(s), 9.8*Unit(m/s^2)):
> numelems(sys);
```

$$4004$$

- The **TestDimensions** command without further options verifies that this is a dimensionally consistent set of equations.

```
> Units:-TestDimensions(sys);
```

$$true$$

- With the option below, it computes the dimensions of x_0 and v_{2000}.

```
> Units:-TestDimensions(sys, output=units(x[0], v[2000]));
```

$$\left\{v_{2000}::\left(\frac{\mathrm{m}}{\mathrm{s}}\right), x_0::\mathrm{m}\right\}$$

- This example is about 200 times faster in Maple 2023 than before.
- The Units:-Simple package has been optimized to improve performance of base operations. In order to provide automatic units simplifications and verify consistency, the Units:-Simple package overloads core operations such as add, multiply and exponentiations. In previous versions this could add significant overhead even when calculating with expressions that do not contain units. In Maple 2023 this overhead has been greatly reduced.
- For example, the following example computes a numerical approximation to pi using a Monte Carlo method.

```
> restart;
> tt := time[real]():
  with(Units[Simple]):
  N := 10^6:
  with(LinearAlgebra):
  X := RandomVector(N, generator = 0 .. 1.0, datatype = float[8]):
  Y := RandomVector(N, generator = 0 .. 1.0, datatype = float[8]):
  total := Vector(N, i -> ifelse(X[i]^2 + Y[i]^2 < 1, 1, 0), datatype=
  float[8]):
  add(total[i], i = 1..N) * 4 / N;
  time[real]()-tt:
```

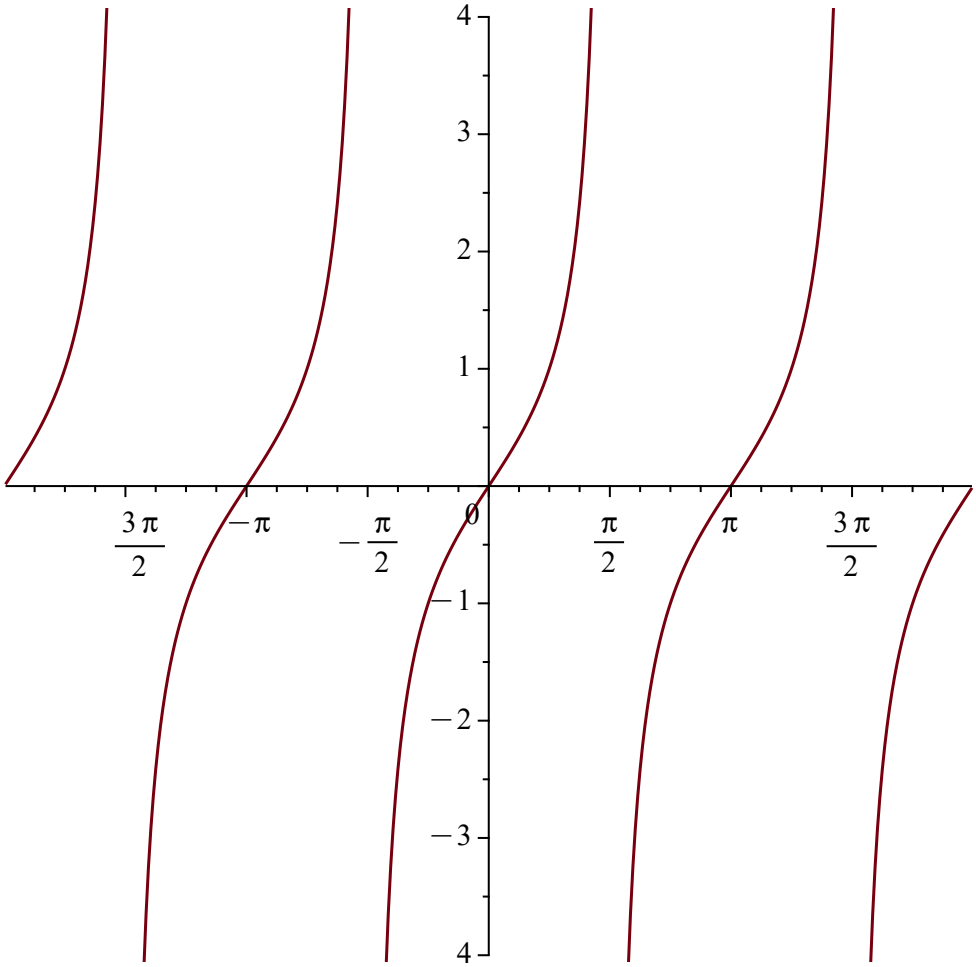
$$3.14037200000000$$

- On a representative computer, this example now completes in 7 seconds, compared to more than 60 seconds in prior versions.

Performance Improvements for Plots

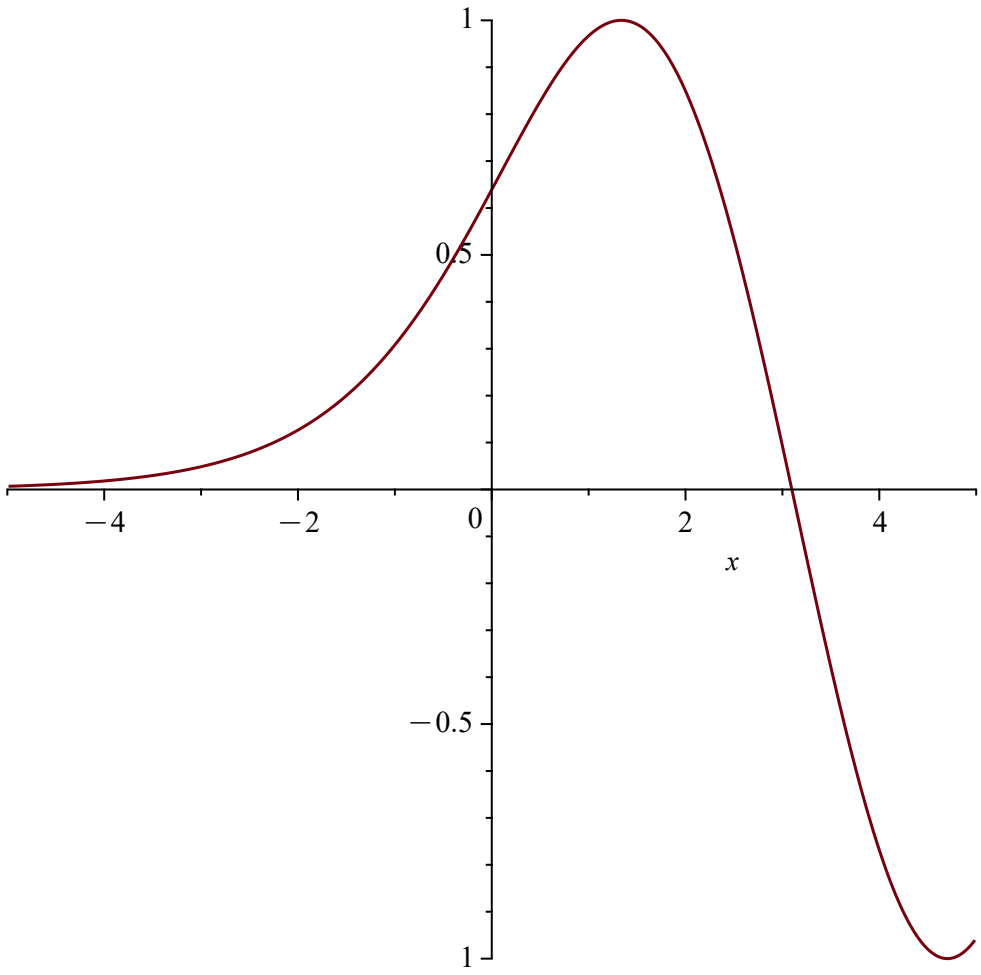
- In Maple 2022, we introduced a new adaptive plotting engine. This engine has been sped up significantly in Maple 2023. For example, the following simple plot has been sped up by about a factor of 4, and it uses about 4 times less memory:

```
> plot(tan);
```



- This example runs about 15 times faster in Maple 2023, and it uses about 10 times less memory.

```
> plot(sin(ln(exp(x) + 1)), x=-5..5);
```



Faster Water and Steam Properties

- The ThermophysicalData:-CoolProp subpackage has been updated to CoolProp 6.4.1. Most notably, the update introduces IAPWS-IF97 water and steam properties--these are faster to compute than those given by IAPWS-95 (which is used by the default Helmholtz equation of state). This is at the expense of slightly less accuracy and applicability over a smaller envelope of temperatures and pressures.
- IAPWS-IF97 is recommended when water properties need to be computed quickly and continuously, such as in the simulation and optimization of steam power cycles.
- Specific heat capacity of water using IAPW-IF97

```
> ThermophysicalData:-CoolProp:-Property(C, "IF97::Water",
  temperature=500*Unit('K'), pressure=1*Unit('atm'));
```

$$1981.542297 \ \frac{\mathrm{J}}{\mathrm{kg\,K}}$$

- Specific heat capacity of water using the Helmholtz equation of state (which can be specified by using "HEOS::Water" or "Water" as the fluid).

```
> ThermophysicalData:-CoolProp:-Property(C, "HEOS::Water",
  temperature=500*Unit('K'), pressure=Unit('atm'));
```

$$1981.609716 \frac{\mathrm{J}}{\mathrm{kg\,K}}$$

- Evaluate the specific heat capacity of water N times using IAPW-IF97 and the Helmholtz equation of state

```
> N:=4000:
```

```
> tt:=time[real]():
  for i to N do
     ThermophysicalData:-CoolProp:-Property(C, "IF97::Water",
  pressure=1*Units:-Unit(atm), enthalpy=Unit(('kJ'/'kg'))):
  end do:
  time[real]()-tt;
```

$$8.359$$

```
> tt:=time[real]():
  for i to N do
     ThermophysicalData:-CoolProp:-Property(C, "HEOS::Water",
  pressure=1*Units:-Unit(atm), enthalpy=Unit(('kJ'/'kg'))):
  end do:
  time[real]()-tt;
```

$$8.949$$

evalhf Hardware Floating-Point Subsystem

- A call to evalhf evaluates an expression to a numerical value using the floating-point hardware of the underlying system. The evaluation is done in double precision. The argument passed to evalhf must be an expression that evaluates to float[8] or complex [8] values as singletons or in an array, matrix, or vector. In Maple 2023, the evalhf subsystem has been extended to handle all built-in (kernel) procedures that process arbitrary inputs and return hardware values.
- Consider the following example which uses the built-in function, rhs. Evaluation of **rhs** and its arguments will be performed outside of evalhf so the non-hardware range structure returned by rtable_dims can be computed, and subsequently given to **rhs**, which will return a hardware compatible structure back to the evalhf subsystem.

```
> restart;
```

```
> A := Array(1..2,datatype=float[8]):
```

```
> evalhf( rhs(rtable_dims(A,1) ) );
```

$$2.$$

- In this example, the given **loopsum** procedure uses the built-in, numelem, which was formerly not supported by evalhf.

```
> loopsum := proc( A )
    local s := 0;
    local n := numelems(A);
    for local i from 1 to n do
        s := s + A[i];
    end do;
    s;
  end proc:
> v :=  LinearAlgebra:-RandomVector(100,datatype=float[8]):
> evalhf(loopsum(v));
```

410.

- These additions extend the reach of evalhf allowing a wider variety of code to be used inside evalhf.

Importing Data from a File

- The ImportVector command can be used to load data from a variety of different file formats. Performance of importing comma-separated value (CSV) files has been improved on Windows. The following example completes in about half the time it used to.

```
> t:=time[real]():
> data:=ImportVector("C:\\data.csv", source=csv[standard],datatype=
  float[8]):
> time[real]()-t:
```

Bit-Shift Operations

- Two new commands integerdivq2exp and integermul2exp provide what are considered hardware bit-shift operations. They are fast methods for dividing or multiplying an integer to a power of 2. integermul2exp computes **i*2^pow**, and integerdivq2exp calculates **trunc(i / 2^pow)**.

```
> integermul2exp( 10, 4 ) - 10*2^4;
```

0

```
> integerdivq2exp(255,4) - trunc(255/2^4);
```

0