Comparison of Numerical Performance of Mathematica 15 and Maple 2026.1

Summary

While Maple claims support for a significant subset of the numerical computations performed by Mathematica, in most cases, much faster methods have been implemented in Mathematica.

Over a test suite of 800 tasks, covering different operations on different types of numerical data, Mathematica was faster in 779 cases. The median difference measured found Mathematica to be 97 times faster than Maple.

Category	Tests	Median Mathematica speed
Elementary & special functions	105	3,087 times faster
Elementary & special functions (manual type override)	105	838 times faster
Elementary & special functions (sparse data)	35	24,381 times faster
Complex elementary & special functions	105	40 times faster
Complex elementary & special functions (manual type override)	105	511 times faster
Elementary & special functions (50 digits)	105	51 times faster
High-precision function evaluation	13	92 times faster
Real data operations	30	2 times faster
Real data operations (manual type override)	27	73 times faster
Sparse real data operations	8	116,000 times faster
Upper triangular real data operations	9	7 times faster
Complex number data operations	25	3 times faster
Complex number data operations (manual type override)	25	5 times faster
Integer data operations	22	82 times faster
Integer data operations (manual type override)	20	1,014 times faster
Extended-precision data operations (50 digits)	26	9 times faster
Extended-precision data operations (1,000 digits)	26	7 times faster
Random number generation	9	18 times faster
All tests	800	97 times faster

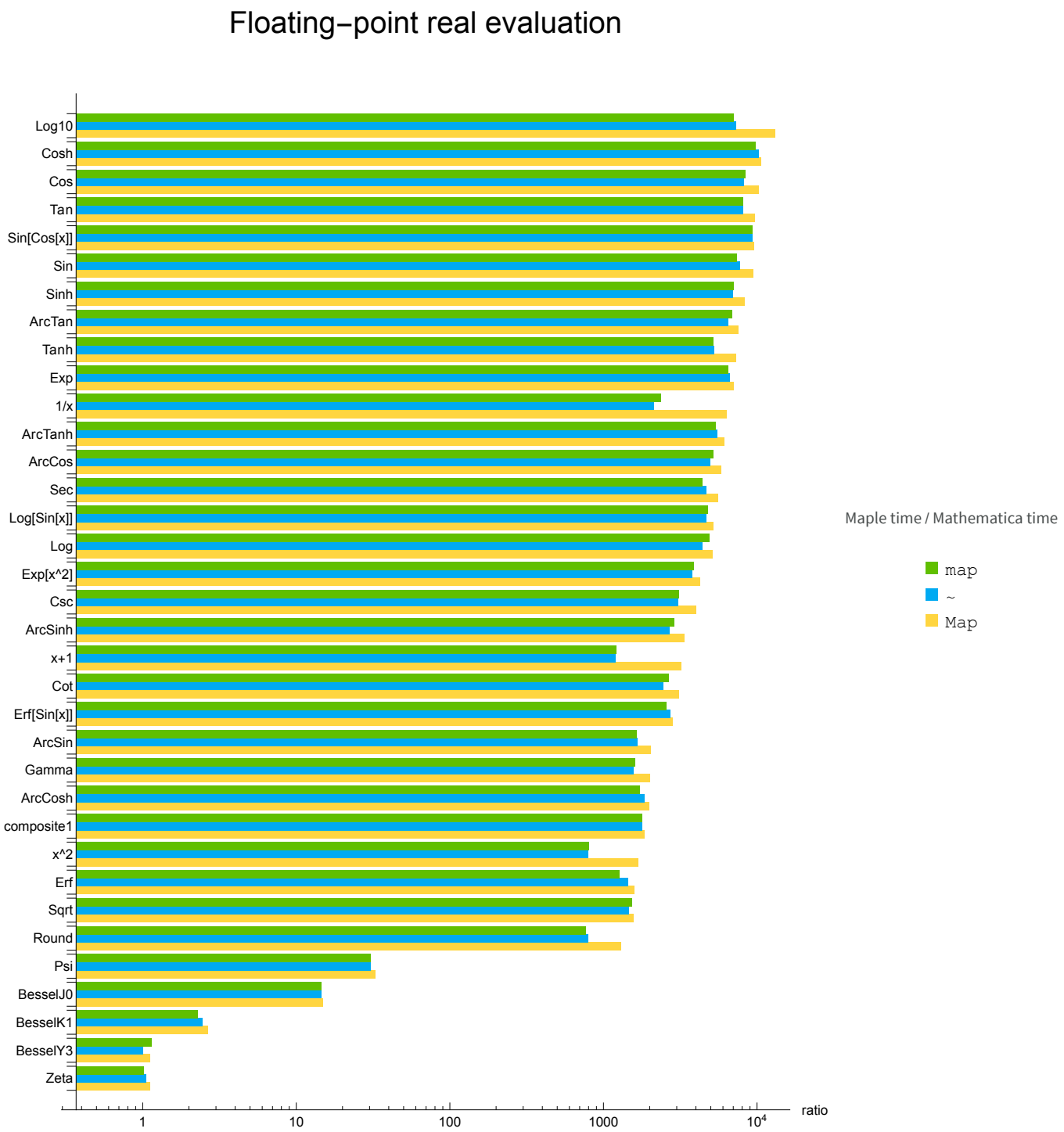
Tests were performed using Mathematica 15 and Maple 2026.1 running on a MacBook Pro with Apple M3 Pro CPU, 36GB RAM running macOS Tahoe 26.5.2.

Function Evaluation

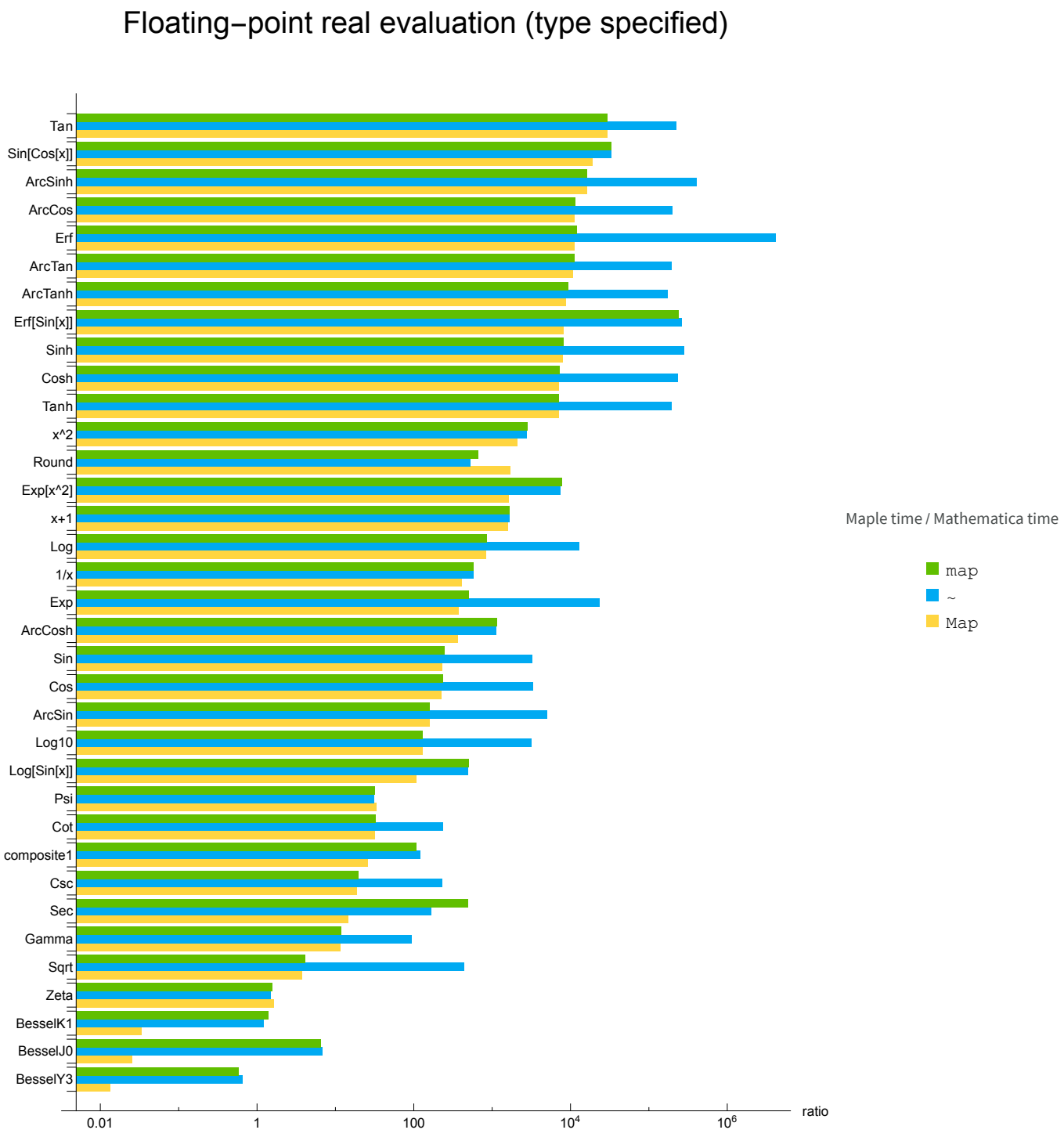
Maple provides a choice of tools for mapping the functions over the data. Different circumstances require different choices and are often a tradeoff between performance and flexibility and code robustness. Maple's `map` and ~ are roughly equivalent to Mathematica's Map command while Maple's Map is a destructive operation (requiiring addiitional work if one wishes to preserve the original data). The timings for all three have been compared to Mathematica's automated mappping.

Reals

For real numbers using the default data type in both systems, Mathematica has a median performance that is 3086.9 times faster than Maple.



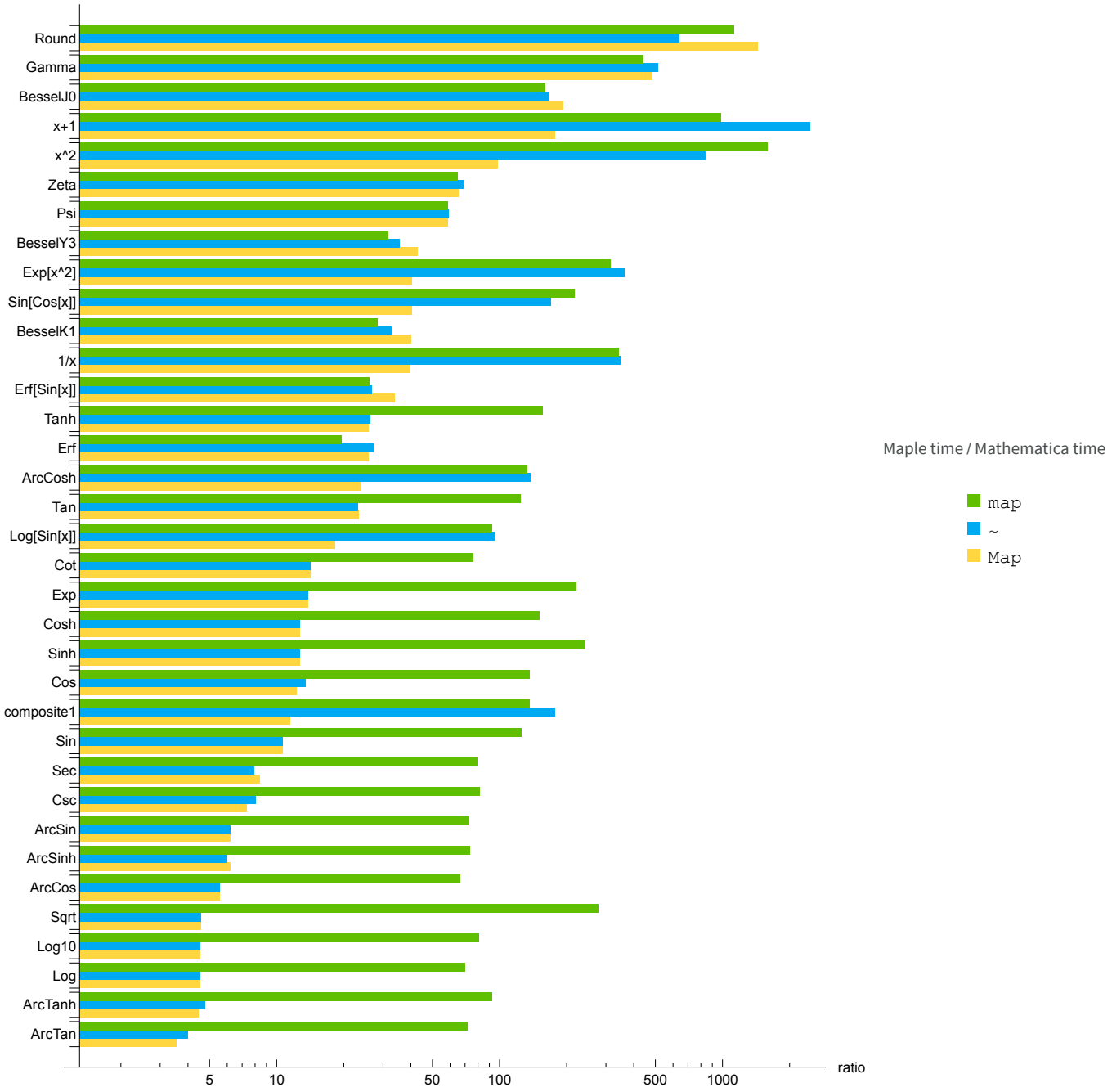
Both systems support optimized data structures that trade better performance for reduced flexibility. The benchmark has been repeated with the data in Maple's `float[8]` data type, and Mathematica's GPUArray structure. With these interventions, the performance gap measures Mathematica as 837.6156815873345 times faster.



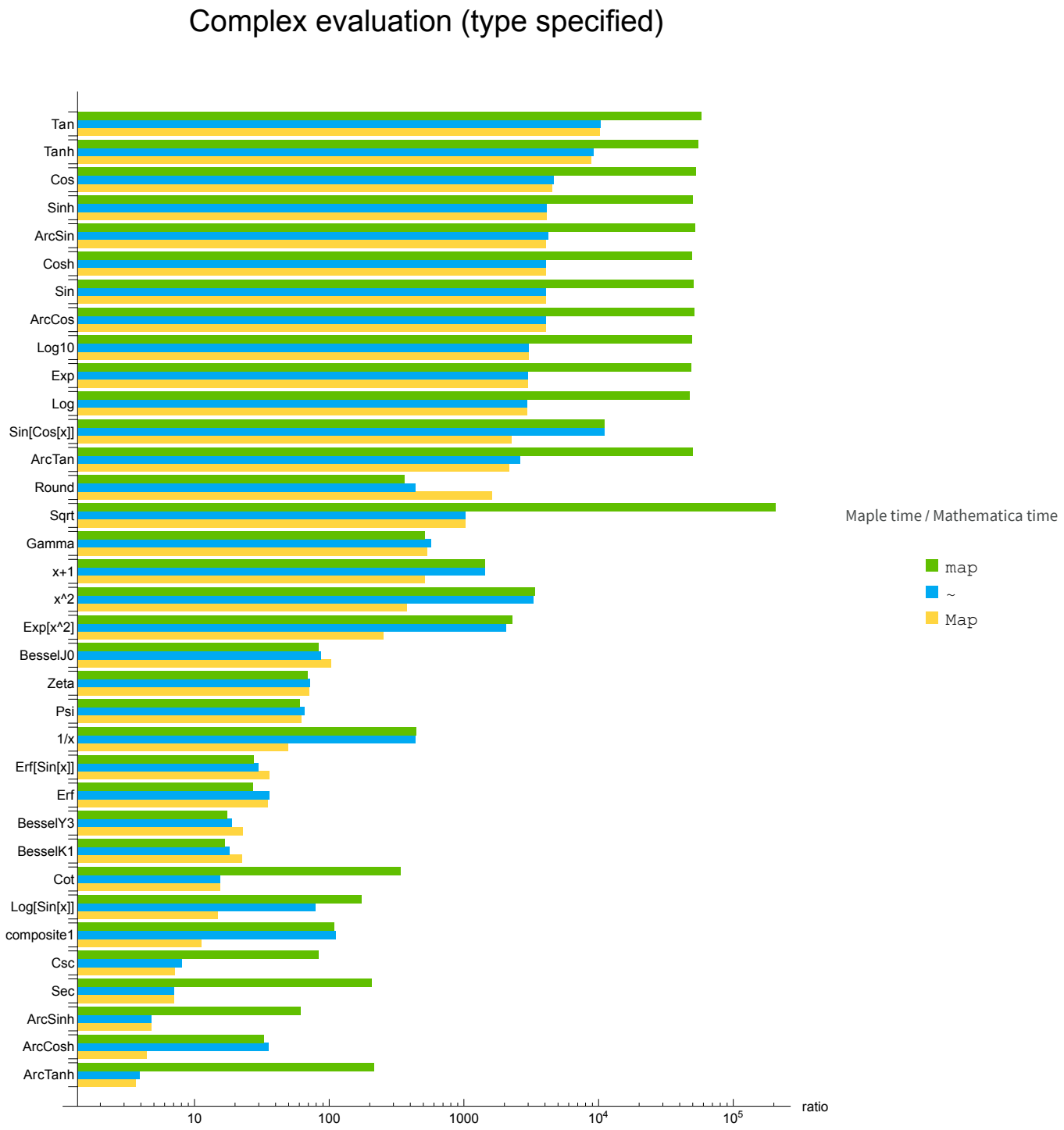
Complexes

For operations on complex numbers, Maple shows a much greater range in relative performance, sometimes only modestly slower than Mathematica when using `Map` or ~ but consistenty much slower when using `map`.

Complex evaluation

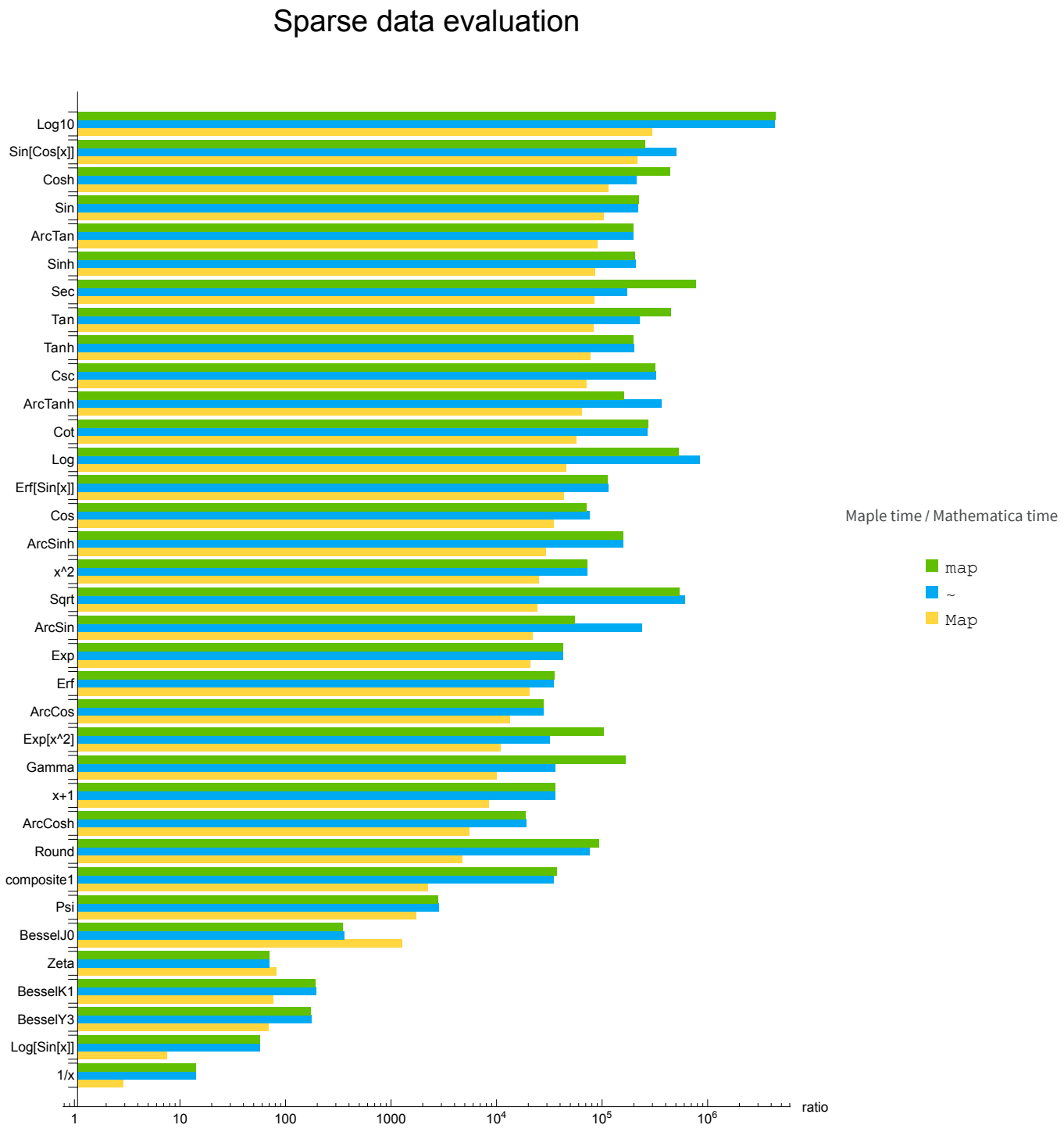


While Maple's `Map` and ~ commands can be modestly accelerated by using its `complex[8]` data type, Mathematica's `GPUArray["Complex"]` type is dramatically faster.



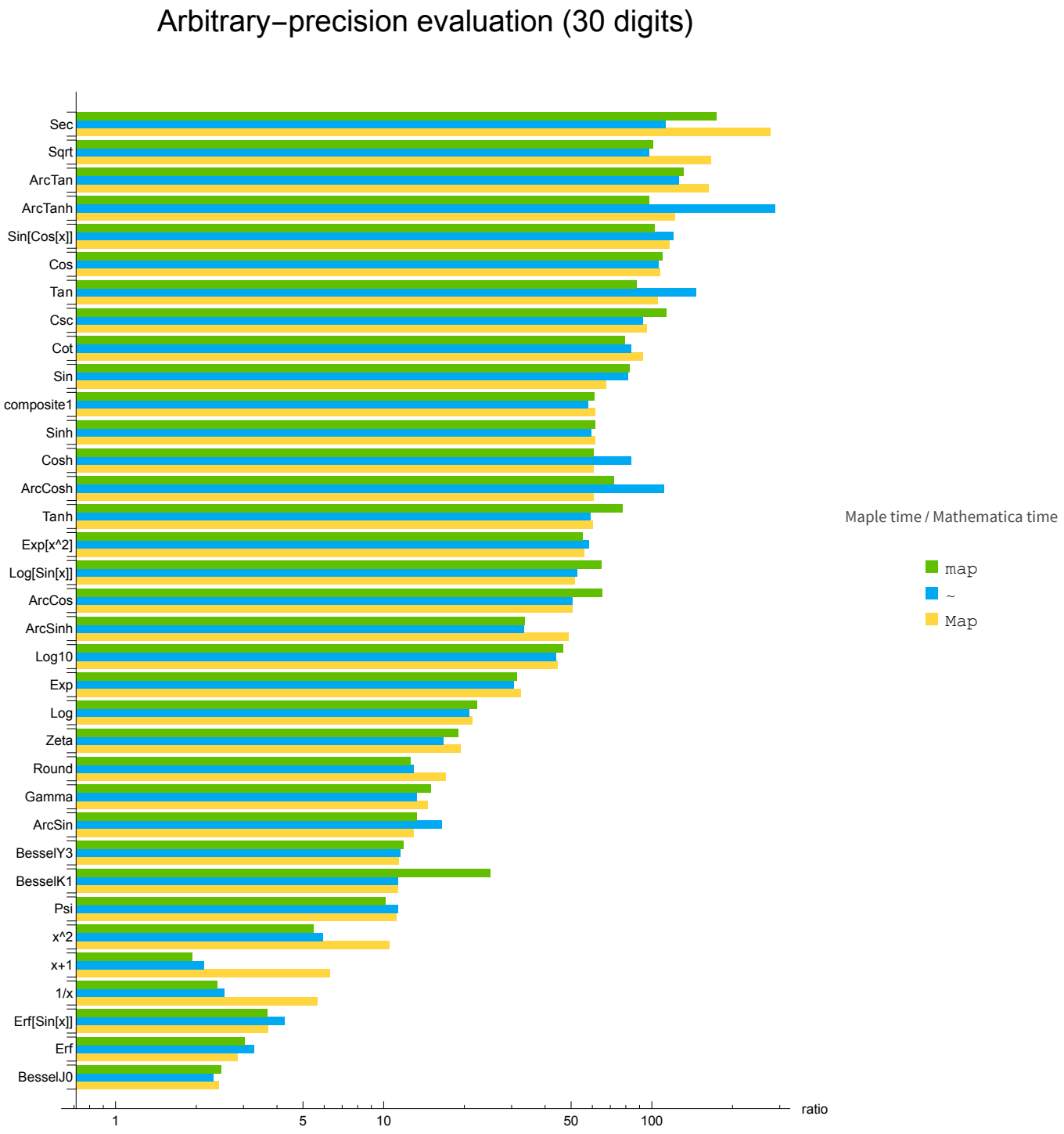
Sparse data

While Maple can represent sparse data efficiently, none of `map`, `Map` or `~` operators apply efficient functions efficiently over sparse data, and only Map yields a sparse data structure as output.

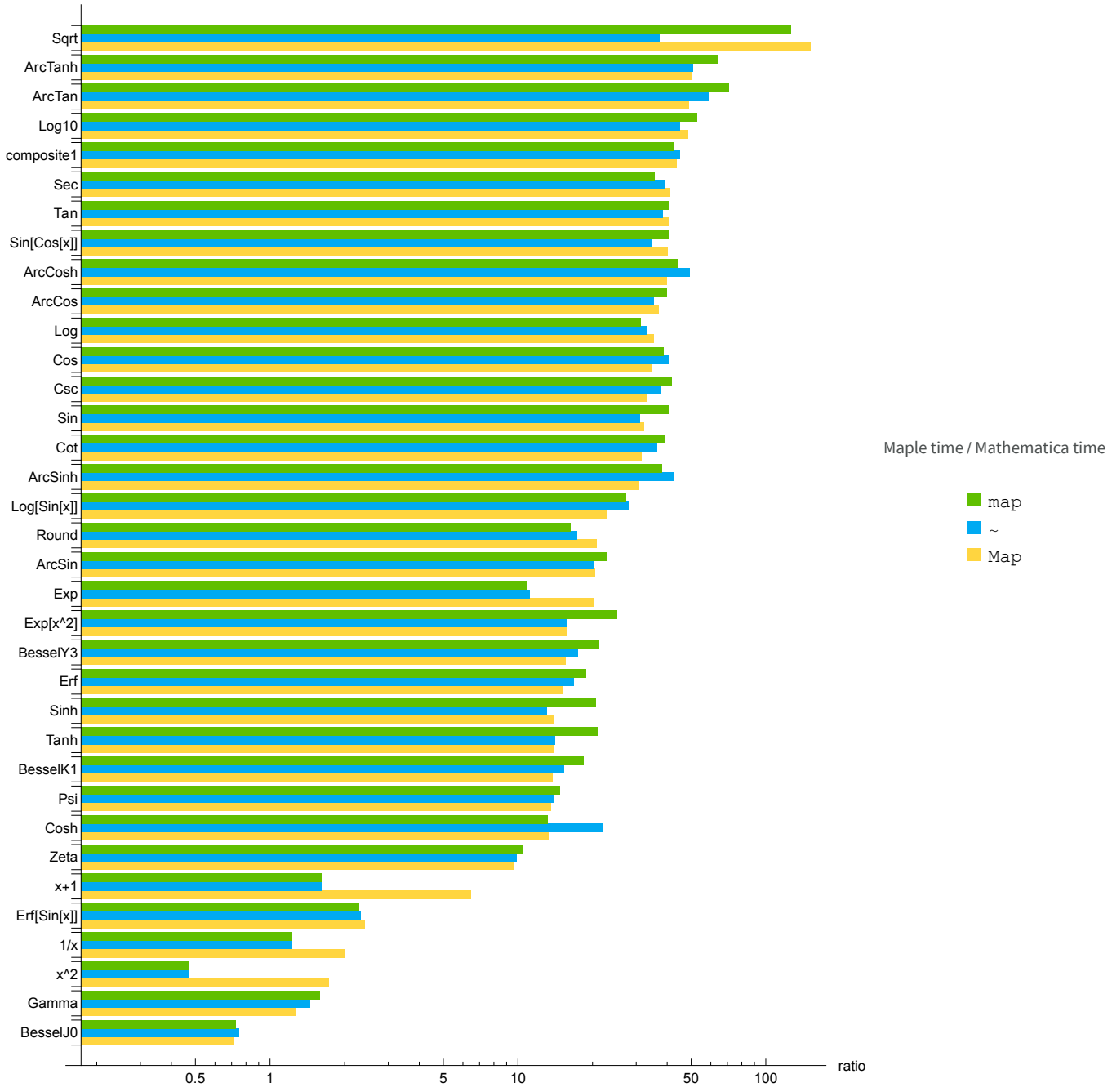


Extended precision

When working in 50-decimal-place extended arithmetic, it makes little difference how you apply functions in Maple. The median performance is 51 times slower than Mathematica.



Working at 1000-decimal-place precision produces similar results.

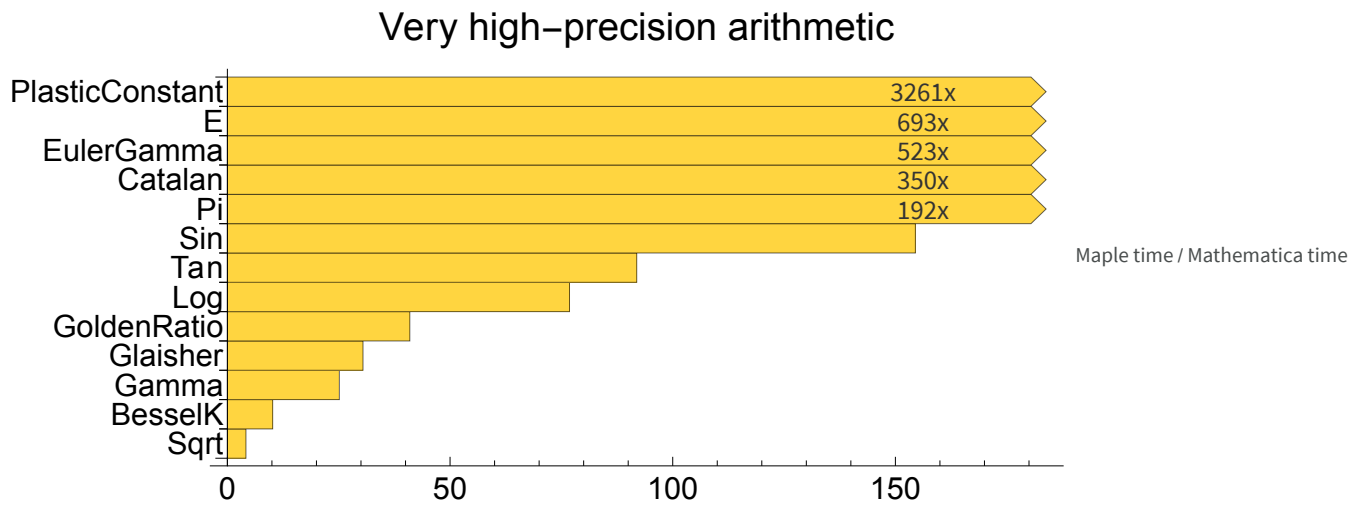


Implementation notes

Tests performed on a vector of 10^5 uniform random numbers in the interval [0,1] for real data tests, [−10,10] for integers and [0,1]+I*[0,1] for complex data. Sparse data tests performed on a vector of 2×10^{10} reals with a sparsity ratio of 1:10000. For Maple `map` and ~ operation over sparse data, the functions were adjusted to offset by 1 since errors such as log(0) abort the entire test, and Maple cannot represent sparse data where the default value is nonzero.

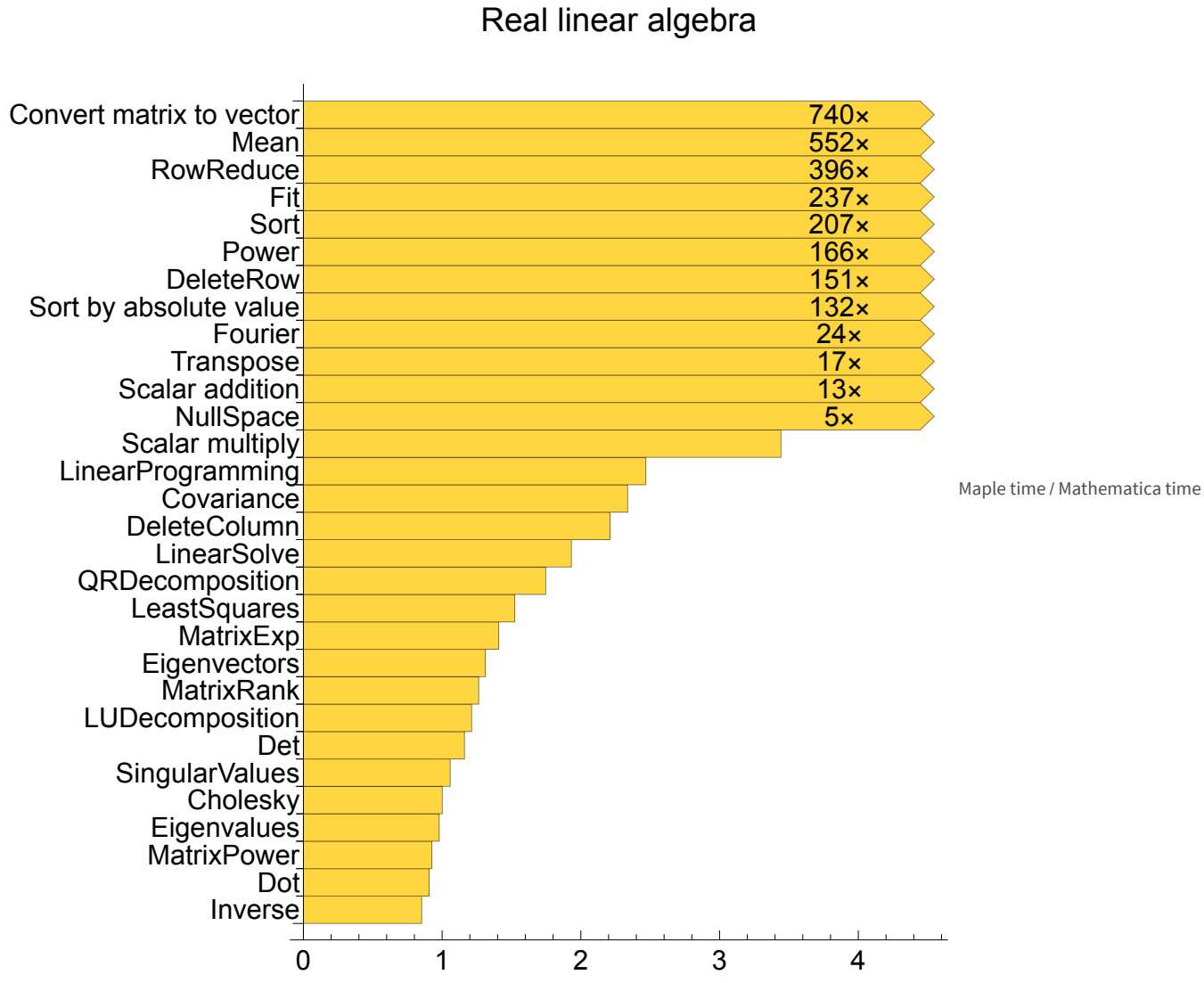
High-Precision Evaluation

When evaluating numeric expressions to very high precision, Mathematica provides automatic precision tracking to ensure that it achieves the target number of correct digits (as opposed to just using input with the target number of digits). The examples in this test were too simple for this to matter, but despite this extra verification work, Mathematica evaluated the following expressions to 1,000,000 digits (`BesselK`, `Gamma` and `Glaisher` to 5,000 digits) with a median performance 92 times faster than Maple.

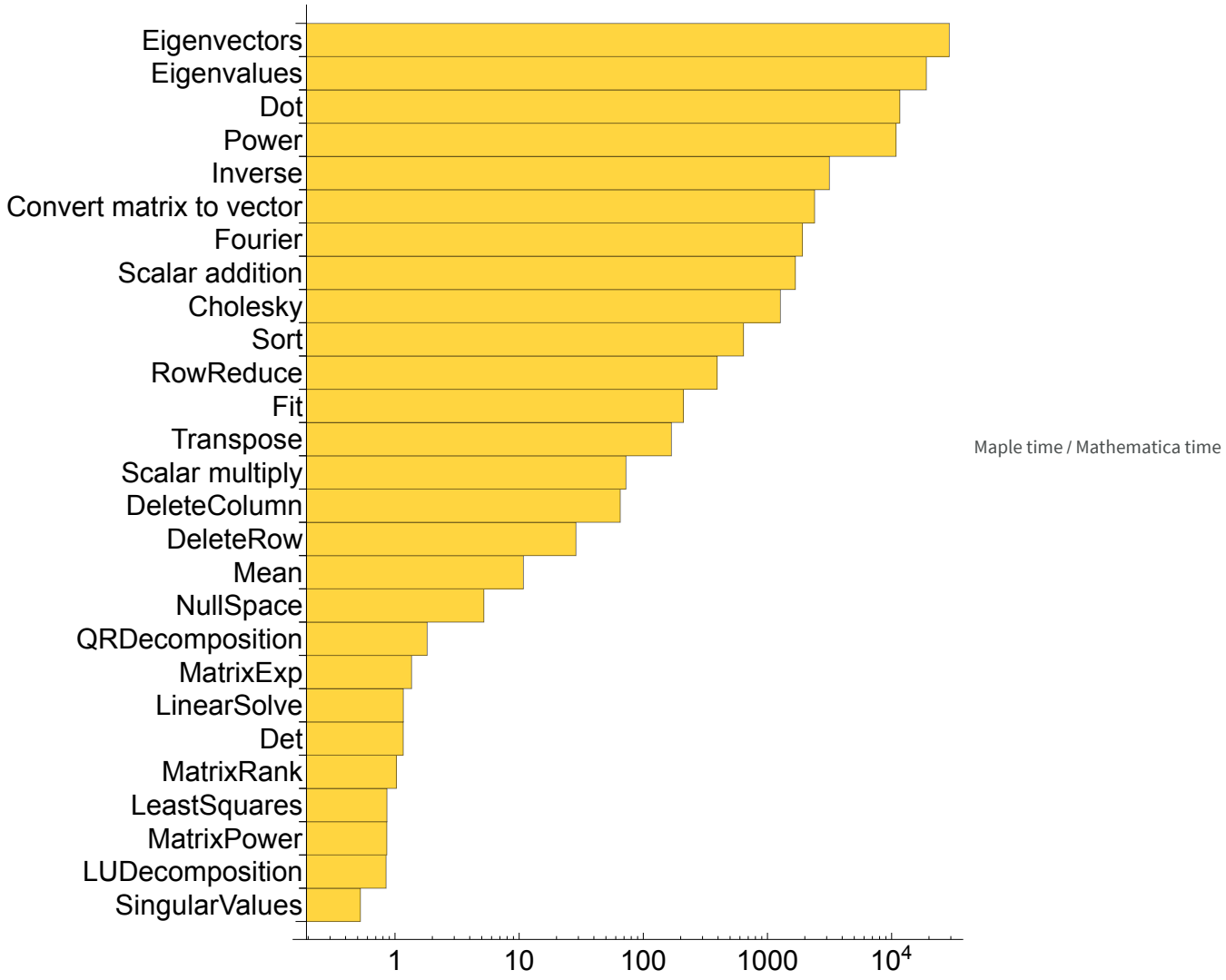


Machine-Precision Real Linear Algebra

While some matrix and list operations have broadly similar performance for real numbers using default data types, many are signifcantly slower. Mathematica operations are 73 times faster than Maple with manual-type control and 2 times faster using defaults.

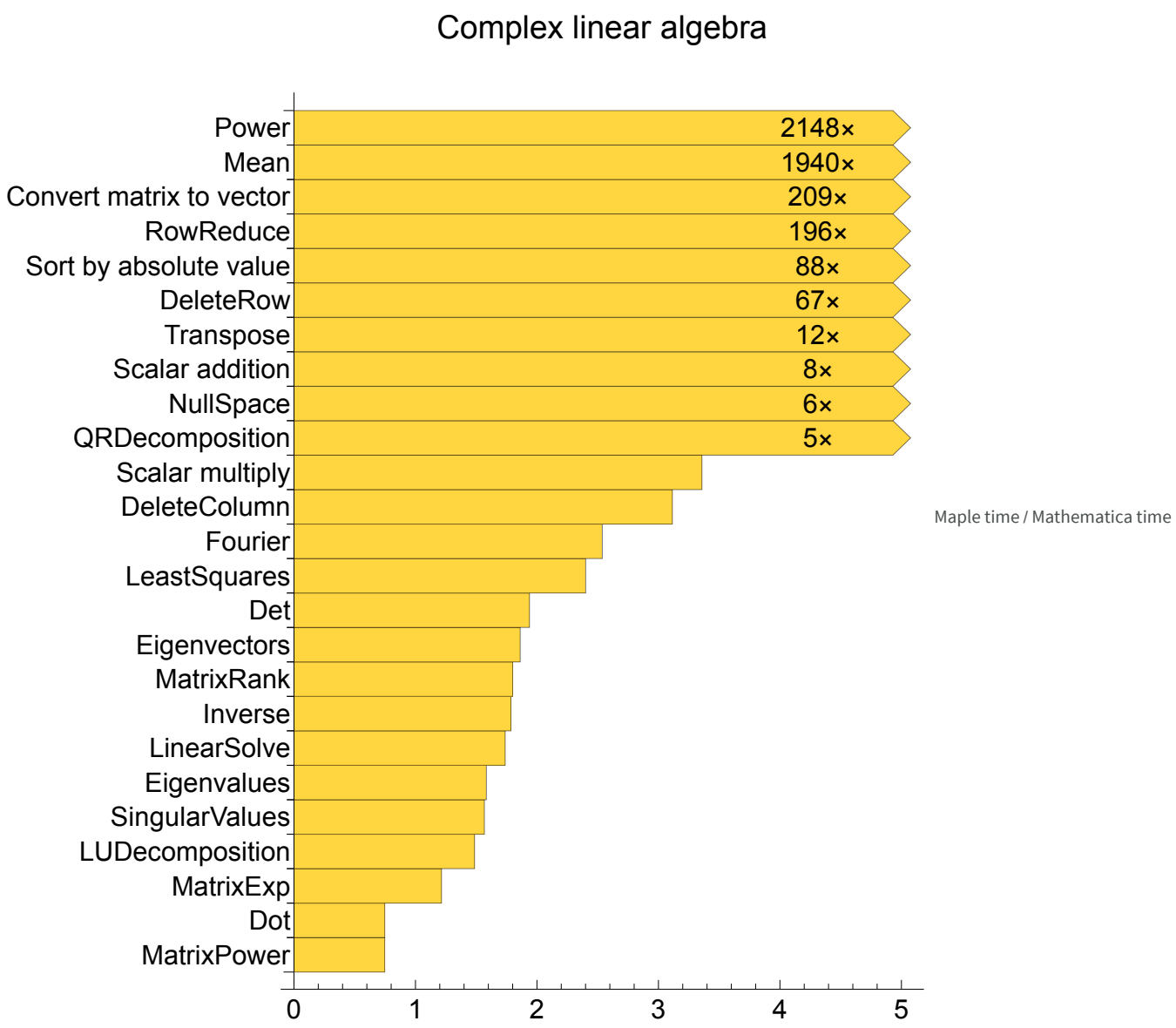


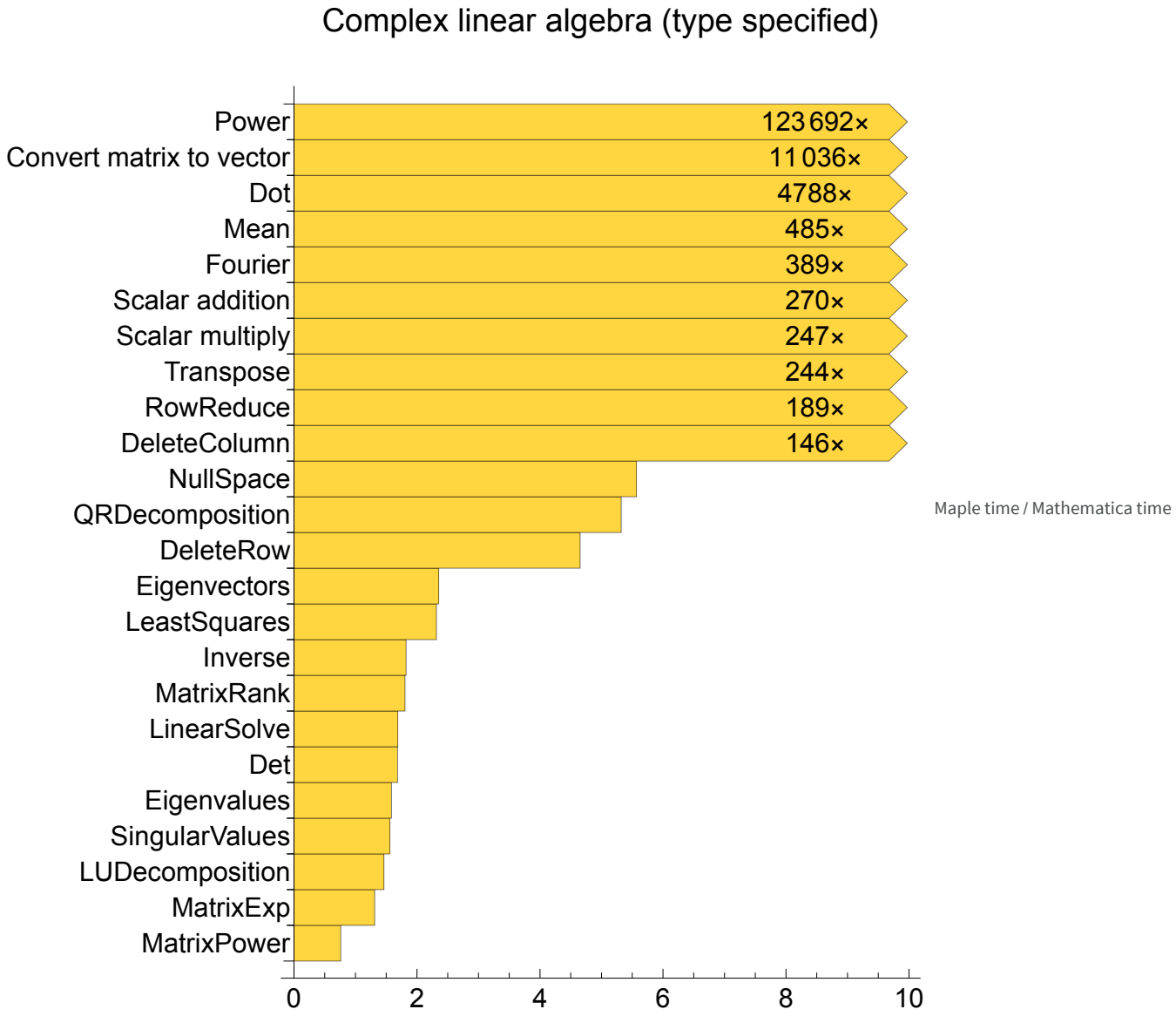
When using manual-type control, many Maple operations are dramatically slower than Mathematica.


Real linear algebra (type specified)
Eigenvectors
Eigenvalues
Dot
Power
Inverse
Convert matrix to vector
Fourier
Scalar addition
Cholesky
Sort
RowReduce
Fit
Transpose
Scalar multiply
DeleteColumn
DeleteRow
Mean
NullSpace
QRDecomposition
MatrixExp
LinearSolve
Det
MatrixRank
LeastSquares
MatrixPower
LUDecomposition
SingularValues
1
10
100
1000
10^4
Maple time / Mathematica time

Machine-Precision Complex Linear Algebra

For complex numbers, there is little advantage to manually setting the data type in Maple. Mathematica is more than 3 times faster.





Implementation notes

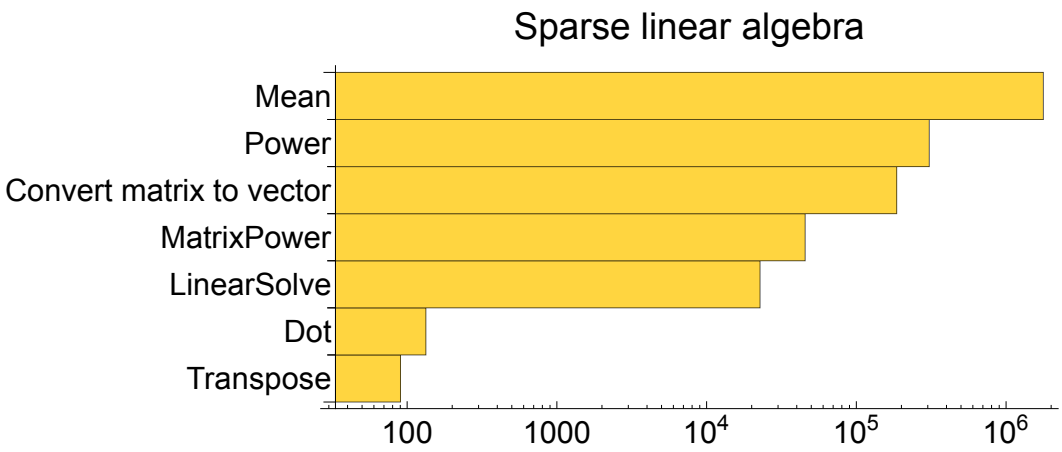
Functions in Maple's `Statistics` packages work only on real numbers. For this reason, the `Student[Statistics]:-Mean` operation has been used.

All other tests were performed using the documented command in each system.

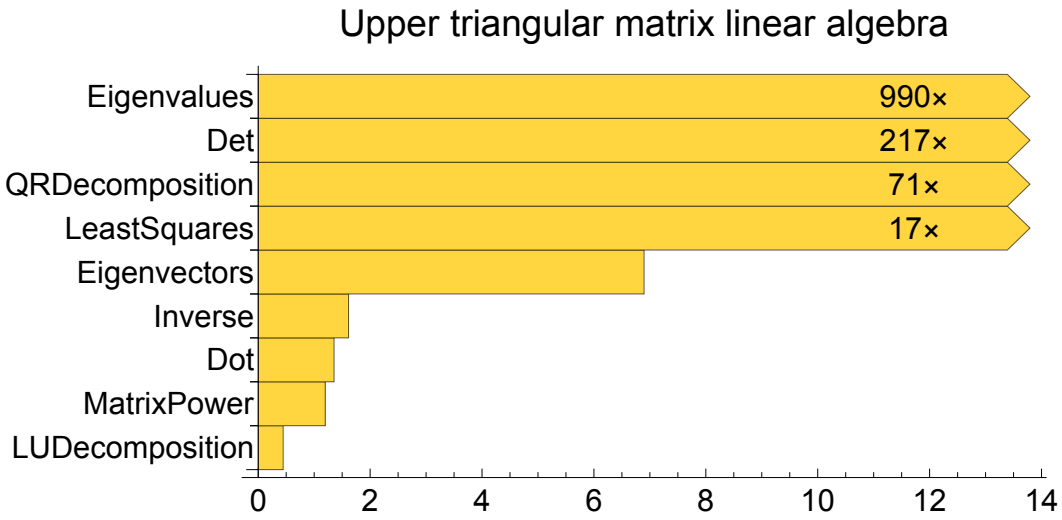
Sparse and Structured Data

Maple and Mathematica both provide sparse data storage; however, Maple appears to have only implemented sparse methods for `LinearSolve` and `Transpose`. During the test, Maple's `LinearSolve` command generates a large number of messages stating, "Warning, solution did not converge within 200 iterations," suggesting that it may not have completed the task for some problem sizes.

Note that data is shown on a log scale, since the difference in performance is too great to display on a linear scale.



Some linear algebra operations have fast algorithms for partially sparse structured arrays such as upper triangular matrices. These have been implemented in Mathematica, giving it a greater speed advantage for these common cases.

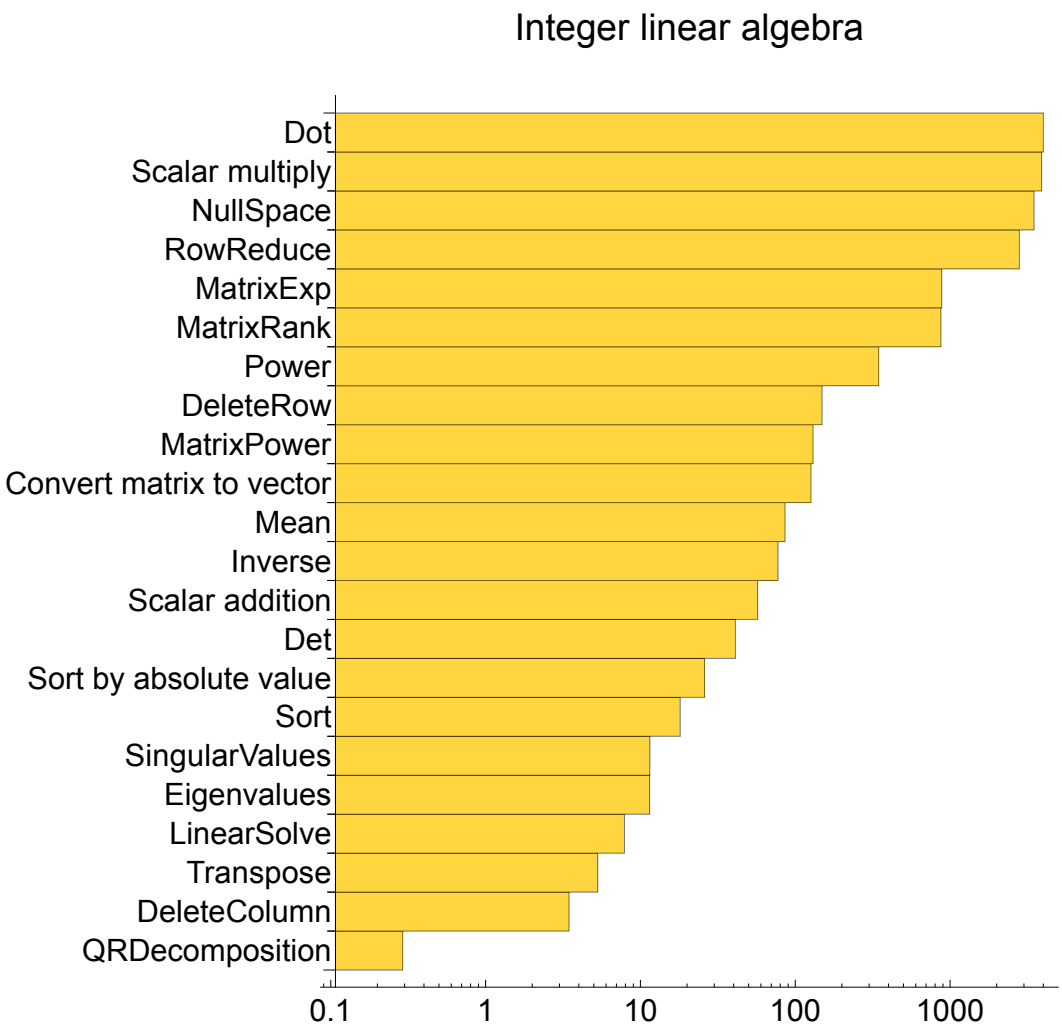


Implementation notes

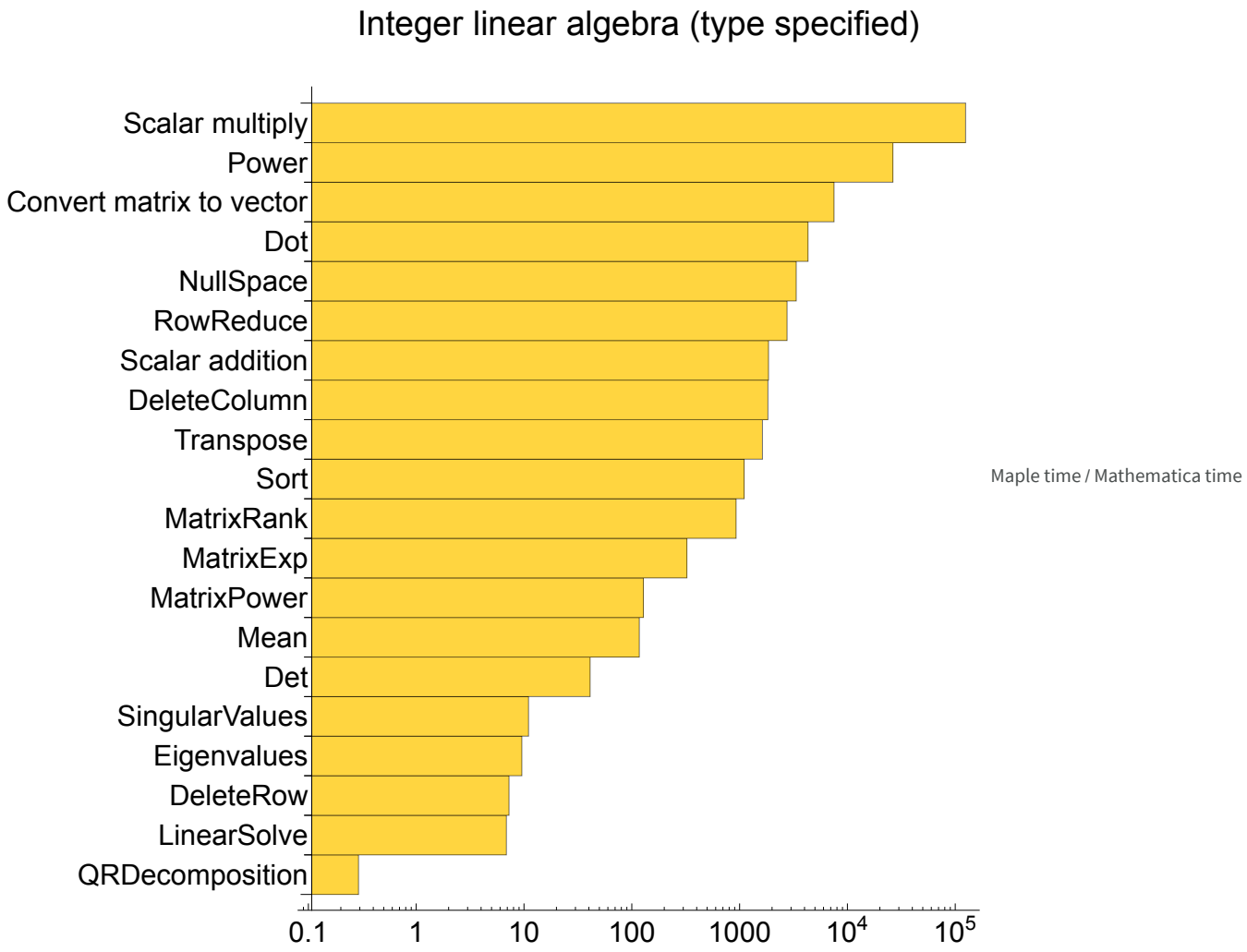
Tests for which neither system has sparse or structured data methods have been omitted, since these would be a repeat of the dense tests above.

Integer Linear Algebra

Both Mathematica and Maple can perform exact integer arithmetic; however, again the Maple implementation does not seem to include many optimized integer algorithms. One of the biggest differences is seen with perhaps the most important matrix operation—multiplication (`Dot`)—where Mathematica was more than 4,300 times faster.



When using Maple's `integer[8]` and Mathematica's `GPUArray["Integer"]` data types, the difference increases dramatically from Mathematica being 82 times faster to 1014 times faster.

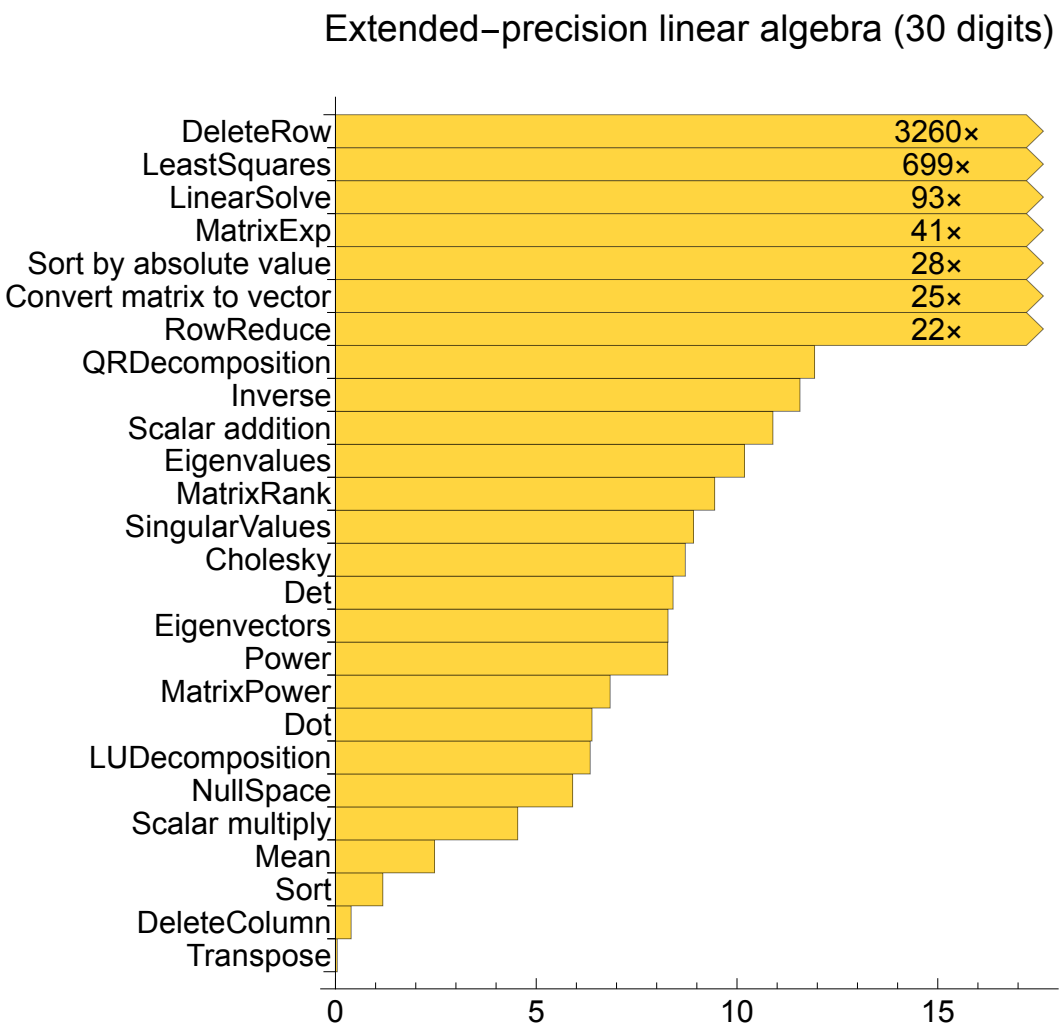


Implementation notes

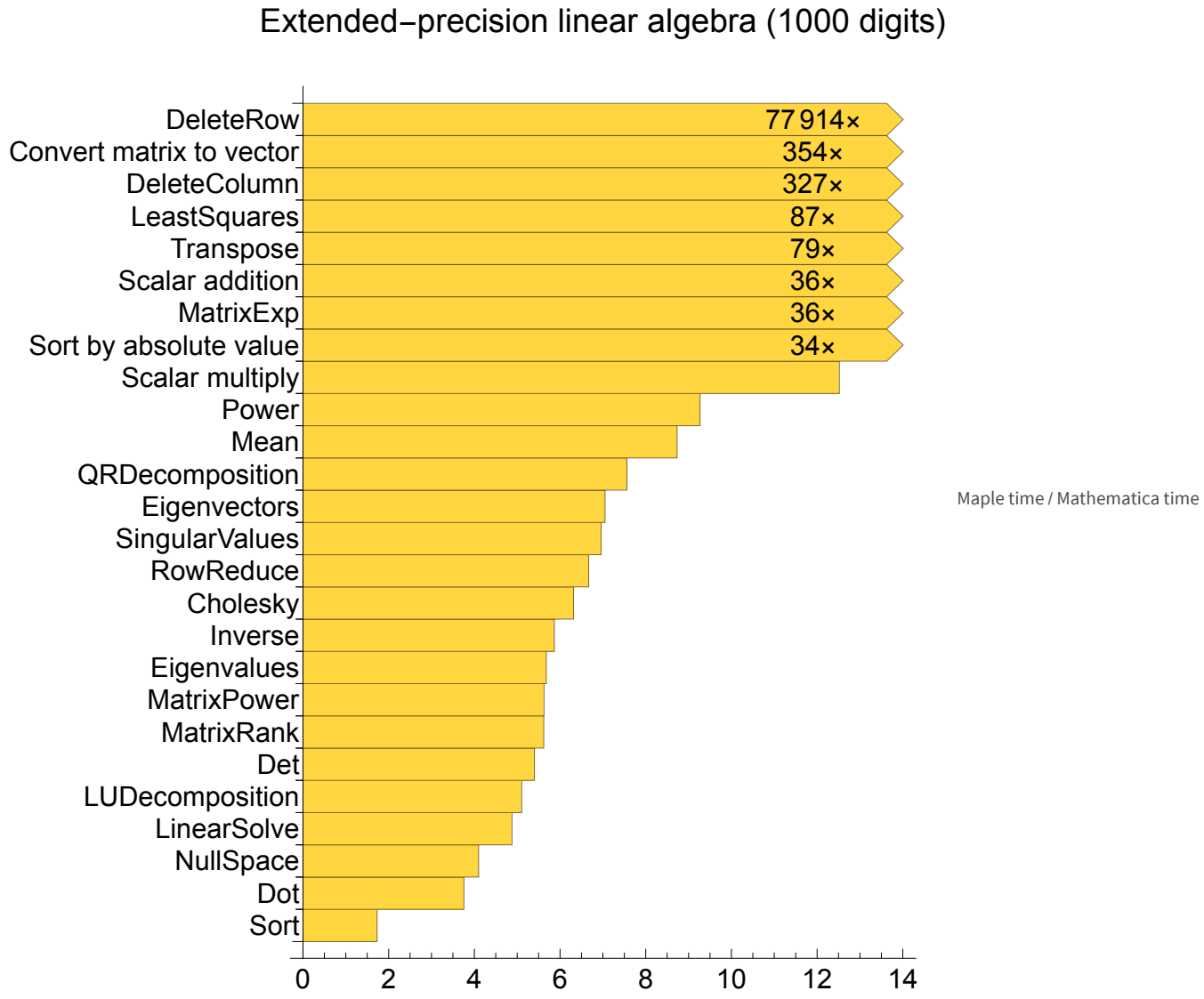
Functions in Maple's `Statistics` packages work only on real numbers. For this reason, the `Student[Statistics]:-Mean` operation has been used and the `Fit` test has been omitted.

Extended-Precision Data

Both Mathematica and Maple handle arbitrary-precision arithmetic. Only Mathematica tracks the number of reliable digits in the results of such calculations, and yet despite doing this additional validation work, Mathematica was 51 times faster for 50-decimal-place matrix computations.



At 1,000-digit precision, much more of the work is done by the GMP library, but Maple is still 7 times slower than Mathematica.



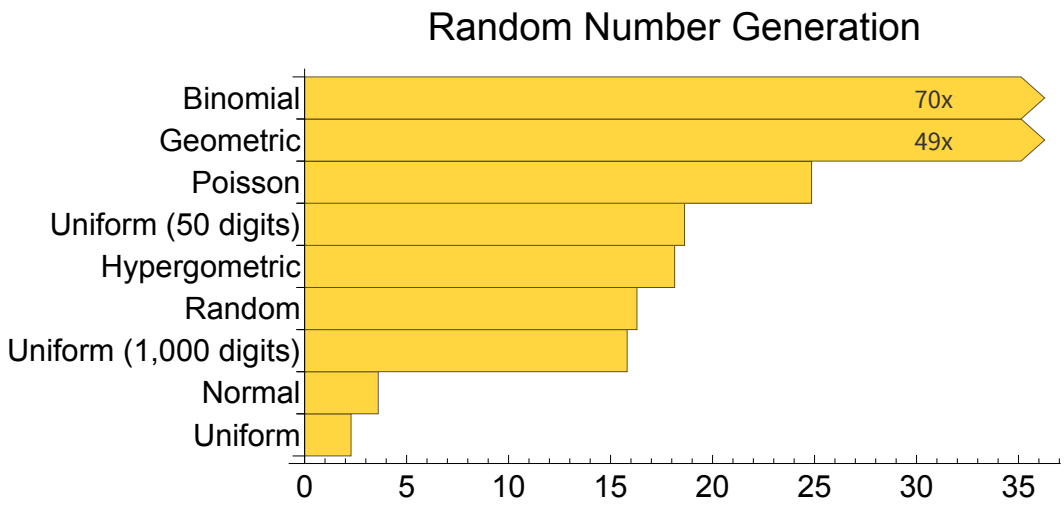
Implementation notes

Some Maple functions are limited to machine-precision numbers. For this reason, tests for `Fit` and `Fourier` have been omitted and `Mean` has been implemented using `add`.

Random Numbers

Median performance for random number generation shows Mathematica to be more than 18 times faster than Maple.

Matrices with 2,000×2,000 elements were generated for each number type using a uniform distribution [−10,10]. Normal, Poisson and binomial distributions were used to create a vector of 10^7 samples.



Implementation notes

Documented Maple commands were used in all cases.
Maple's default type for reals and its special `float[8]` data generation were both compared to the default data type in Mathematica.
Maple's default type for integers and its special `integer[8]` data generation were both compared to the default data type in Mathematica.
Maple's Poisson and binomial distributions (and other discrete distributions) generate real numbers, while Mathematica correctly generates integers.

General Testing Methodology

Tests were performed using a MacBook Pro with Apple M3 Pro CPU, 36GB RAM running macOS Tahoe 26.5.2. Tests were performed using Maple 2026.1 .1 and Mathematica 15 default installation.

Some tests have been scaled back to smaller problems in the Maple version to enable Maple to perform them in the available time and memory. Where standard Maple functions cannot perform the requested computations, tests have been excluded or, where trivial, alternative Maple implementations have been created. See source code for details. Median values are calculated only from successfully completed tests.

For most functions, the test suite finds the average time for five evaluations for each size of problem, but high-precision evaluations were performed only once in each test run, since both systems cache the results for later reuse. The entire test suite is repeated in fresh sessions, and the mean time for each successful test was used.

Relative performance ratios use the largest problem size for each test performed, except where the Maple test has had to be scaled back. In these cases, the available data has been fitted to the curve $a x^2 + b x$, and this model has been used to estimate a time for the largest problem size timed in Mathematica.

Source code for the tests is included so that results can be replicated independently.

Appendix: Test Source Code

Mathematica code

Test code

```
$HistoryLength = 0;
SetDirectory[CreateDirectory[
   FileNameJoin[{"/Users/test/Desktop/BenchmarkData", DateString["ISODateTime"]}]]];
steps = 6;
repeats = 4;
```

```
maxVector = 10^5;
maxMatrix = 2000;
maxSparseMatrix = 20 000;
maxSparseVector = 2 * 10^6;

makeData[type_, i_Real] := makeData[type, Round[i]];
accurateTiming[expr_] := With[{time = AbsoluteTiming[expr][[1]]},
   If[time > 10^-5, time, RepeatedTiming[expr][[1]]]];
SetAttributes[accurateTiming, HoldAll];

makeData[type_, i_Integer] := Switch[type, "RealVector", RandomReal[{0, 1}, i],
   "RealMatrix", RandomReal[1, {i, i}], "UpperTriangularRealMatrix",
   UpperTriangularMatrix[UpperTriangularize[1 + makeData["RealMatrix", i]]],
   "ExtendedMatrix", RandomReal[1, {i, i}, WorkingPrecision → 50],
   "ExtendedMatrix1000", RandomReal[1, {i, i}, WorkingPrecision → 1000],
   "ExtendedVector", RandomReal[1, {i}, WorkingPrecision → 50], "ExtendedVector1000",
   RandomReal[1, {i}, WorkingPrecision → 1000], "IntegerMatrix",
   RandomInteger[100, {i, i}], "IntegerVector", RandomInteger[100, i], "SparseMatrix",
   SparseArray[Table[{RandomInteger[{1, i}], RandomInteger[{1, i}]} → Random[],
     {i^2 / 10 000}], {i, i}], "SparseVector",
   SparseArray[Table[RandomInteger[i] → Random[], {Floor[i / 10 000]}], i],
   "ComplexMatrix", RandomComplex[{0, 1 + I}, {i, i}], "ComplexVector",
   RandomComplex[{0, 1 + I}, {i}], "RealMatrixGPU", GPUArray[RandomReal[1, {i, i}]],
   "RealVectorGPU", GPUArray[RandomReal[1, {i}]], "ComplexMatrixGPU",
   GPUArray@RandomComplex[{0, 1 + I}, {i, i}], "ComplexVectorGPU",
   GPUArray@RandomComplex[{0, 1 + I}, {i}], "IntegerVectorGPU",
   GPUArray[RandomInteger[100, i]], "IntegerMatrixGPU",
   GPUArray[RandomInteger[100, {i, i}]], _, Print[type]];

timedReport[path_, fn_, hi_, type_] := Block[{data},
   Export["Mathematica" <> ToString[$VersionNumber] <> Echo[path] <> ".txt",
    Table[{Floor[t], Mean[Table[data = makeData[type, Floor[t]];
        accurateTiming[fn[data]], {repeats}]]},
     {t, hi / steps, hi, hi / steps}], "Table"]];

numericFnTest[fnRow_, n_, type_] := Block[{data, label, fn, scale}, label = fnRow[[1]];
   fn = fnRow[[2]];
   scale = fnRow[[3]];
   data = makeData[type, n * scale];
   {label, accurateTiming[fn[data]]}];

NumericTest[path_, fns_List, n_, type_] :=
  Export["Mathematica" <> ToString[$VersionNumber] <> Echo[path] <> ".txt",
   Table[numericFnTest[fns[[i]], n, type], {i, Length[fns]}], "Table"];
evaluateTest[path_, rows_List] :=
  Export["Mathematica" <> ToString[$VersionNumber] <> path <> ".txt",
   {#[[1]], Apply[accurateTiming, #[[2]]]} & /@ rows, "Table"];

fnList = {{"sqrt", Sqrt, 1}, {"sin", Sin, 1}, {"cos", Cos, 1},
```

```
    {"tan", Tan, 1}, {"arcsin", ArcSin, 1}, {"arccos", ArcCos, 1},
    {"arctan", ArcTan, 1}, {"sec", Sec, 1}, {"csc", Csc, 1}, {"cot", Cot, 1},
    {"exp", Exp, 1}, {"sinh", Sinh, 1}, {"cosh", Cosh, 1}, {"tanh", Tanh, 1},
    {"log", Log, 1}, {"log10", Log10, 1}, {"erf", Erf, 1}, {"GAMMA", Gamma, 0.5},
    {"BesselJ0", BesselJ[0, #] &, 0.5}, {"BesselK1", BesselK[1, #] &, 0.5},
    {"BesselY3", BesselY[3, #] &, 0.5}, {"arcsinh", ArcSinh, 1},
    {"arccosh1p", ArcCosh[1 + #] &, 1}, {"arctanh", ArcTanh, 1}, {"Zeta", Zeta, 1},
    {"composite1", # Sin[Log[#] + 2 * Sqrt[#] + 3] + Sqrt[#] + Log[#] &, 1},
    {"sin_cos", Sin[Cos[#]] &, 1}, {"log_sinh", # Log[Sinh[#]] &, 1},
    {"erf_sin", Erf[Sin[#]] &, 1}, {"exp_x2", Exp[#^2] &, 1},
    {"Psi", PolyGamma, 1}, {"x_plus_1", 1 + # &, 1}, {"inv_x", 1 / # &, 1},
    {"x2", #^2 &, 1}, {"frem_0p5", Mod[#, 0.5] &, 1}};

timedReport[path_, fn_, hi_, type_] := Block[{data},
   Export["Mathematica" <> ToString[$VersionNumber] <> path <> ".txt", Table[{Floor[t],
      Mean[Table[
        data = makeData[type, Floor[t]];
        AbsoluteTiming[fn[data]][[1]], {repeats}]]}
     , {t, hi/steps, hi, hi/steps}], "Table"]];


highPrecisionReport[path_, fns_, n_] :=
  Export["Mathematica" <> ToString[$VersionNumber] <> path <> ".txt",
   Map[AbsoluteTiming[N[#, n]][[1]] &, fns]];
NumericTest[path_, fns_List, n_, type_] :=
  Block[{data}, Export["Mathematica" <> ToString[$VersionNumber] <> path <> ".txt",
    Table[data = makeData[type, n];
     AbsoluteTiming[i[data]][[1]], {i, fns}]]];

evaluateTest[path_, expr_Hold] :=
 Export["Mathematica" <> ToString[$VersionNumber] <> path <> ".txt",
  Apply[List, First /@ Map[AbsoluteTiming, expr]]]
```

In[ ]:=

```
NumericTest["ElementaryFunctionsReal", fnList, maxVector, "RealVector"];
NumericTest["ElementaryFunctionsComplex", fnList, maxVector, "ComplexVector"];
NumericTest["ElementaryFunctionsRealGPU", fnList, 10 * maxVector, "RealVectorGPU"];
NumericTest["ElementaryFunctionsComplexGPU", fnList, maxVector, "ComplexVectorGPU"];
NumericTest["ElementaryFunctionsSparse", fnList, maxSparseVector, "SparseVector"];
NumericTest["ElementaryFunctionsExtended", fnList, maxVector / 10, "ExtendedVector"];
NumericTest["ElementaryFunctionsExtended1000",
  fnList, maxVector / 10, "ExtendedVector1000"];

timedReport["FourierReal", Fourier, 10 * maxVector, "RealVector"];
timedReport["SortReal", Sort, maxVector, "RealVector"];
timedReport["SortCustomReal", SortBy[#, Abs] &, maxVector, "RealVector"];
timedReport["MeanReal", Mean, 10 * maxVector, "RealVector"];
timedReport["DotReal", #.# &, 2 * maxMatrix, "RealMatrix"];
timedReport["InverseReal", Inverse, 2 * maxMatrix, "RealMatrix"];
timedReport["LinearSolveReal",
  LinearSolve[#, Sort[#[[All, 1]]] &, 2 * maxMatrix, "RealMatrix"];
```

```
timedReport["CholeskyReal",
  CholeskyDecomposition[Transpose[#].#] &, maxMatrix, "RealMatrix"];
timedReport["MatrixPowerReal", MatrixPower[#, 5] &, maxMatrix, "RealMatrix"];
timedReport["DetReal", Det, 2 * maxMatrix, "RealMatrix"];
timedReport["TransposeReal", Transpose, maxMatrix, "RealMatrix"];
timedReport["FlattenReal", Flatten, maxMatrix, "RealMatrix"];
timedReport["EigenvaluesReal", Eigenvalues, maxMatrix, "RealMatrix"];
timedReport["EigenvectorsReal", Eigenvectors, maxMatrix, "RealMatrix"];
timedReport["LinearProgrammingReal",
  LinearProgramming[Abs[#[[1]]], Abs[#], Abs[#[[2]]], Method → "CLP"] &,
  maxMatrix, "RealMatrix"];
timedReport["ElementPowerReal", #^5 &, maxMatrix, "RealMatrix"];
timedReport["MovingAverageReal", MovingAverage[#, 10] &, maxVector, "RealVector"];
timedReport["FitReal", (Function[{data}, Fit[data, x, x]][Transpose[{#, #}]]) &,
  4 * maxVector, "RealVector"];
timedReport["MatrixExpReal", MatrixExp, maxMatrix / 5, "RealMatrix"];
timedReport["CovarianceReal", Covariance, maxMatrix, "RealMatrix"];
timedReport["MatrixRankReal", MatrixRank, 3 maxMatrix, "RealMatrix"];
timedReport["LeastSquaresReal",
  LeastSquares[#, Sort[#[[All, 1]]] &, 2 * maxMatrix, "RealMatrix"];
timedReport["LUDecompositionReal",
  LUDecomposition[#, TargetStructure → "Classic"] &, 2 * maxMatrix, "RealMatrix"];
timedReport["SingularValuesReal", SingularValueList, 2 * maxMatrix, "RealMatrix"];
timedReport["QRDecompositionReal", QRDecomposition, maxMatrix, "RealMatrix"];
timedReport["AddOneReal", # + 1 &, maxMatrix, "RealMatrix"];

timedReport["FourierComplex", Fourier, 4 * maxVector, "ComplexVector"];
timedReport["SortCustomComplex", SortBy[#, Abs] &, maxVector, "ComplexVector"];
timedReport["MeanComplex", Mean, 10 maxVector, "ComplexVector"];
timedReport["DotComplex", #.# &, maxMatrix, "ComplexMatrix"];
timedReport["InverseComplex", Inverse, maxMatrix, "ComplexMatrix"];
timedReport["LinearSolveComplex",
  LinearSolve[#, Sort[#[[All, 1]]] &, maxMatrix, "ComplexMatrix"];
timedReport["CholeskyComplex",
  CholeskyDecomposition[ConjugateTranspose[#].#] &, maxMatrix, "ComplexMatrix"];
timedReport["MatrixPowerComplex", MatrixPower[#, 5] &, maxMatrix / 2, "ComplexMatrix"];
timedReport["DetComplex", Det, maxMatrix, "ComplexMatrix"];
timedReport["EigenvaluesComplex", Eigenvalues, maxMatrix, "ComplexMatrix"];
timedReport["EigenvectorsComplex", Eigenvectors, maxMatrix, "ComplexMatrix"];
timedReport["TransposeComplex", Transpose, maxMatrix, "ComplexMatrix"];
timedReport["FlattenComplex", Flatten, maxMatrix, "ComplexMatrix"];
timedReport["ElementPowerComplex", #^5 &, maxMatrix, "ComplexMatrix"];
timedReport["MatrixExpComplex", MatrixExp, maxMatrix / 2, "ComplexMatrix"];
timedReport["MatrixRankComplex", MatrixRank, 3 / 2 maxMatrix, "ComplexMatrix"];
timedReport["LeastSquaresComplex",
  LeastSquares[#, Sort[#[[All, 1]]] &, maxMatrix, "ComplexMatrix"];
timedReport["LUDecompositionComplex",
  LUDecomposition[#, TargetStructure → "Classic"] &, maxMatrix, "ComplexMatrix"];
timedReport["QRDecompositionComplex", QRDecomposition, maxMatrix, "ComplexMatrix"];
```

```
timedReport["SingularValuesComplex", SingularValueList, maxMatrix, "ComplexMatrix"];
timedReport["AddOneComplex", # + 1 &, maxMatrix, "ComplexMatrix"];

timedReport["DetUpperTriangularReal", Det, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["DotUpperTriangularReal", #.# &, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["EigensystemUpperTriangularReal",
  Eigensystem, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["EigenvaluesUpperTriangularReal",
  Eigenvalues, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["EigenvectorsUpperTriangularReal",
  Eigenvectors, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["InverseUpperTriangularReal",
  Inverse, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["LeastSquaresUpperTriangularReal",
  LeastSquares[#, Sort[#[[All, 1]]] &, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["MatrixPowerUpperTriangularReal",
  MatrixPower[#, 5] &, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["QRDecompositionUpperTriangularReal",
  QRDecomposition, maxMatrix, "UpperTriangularRealMatrix"];
timedReport["LUDecompositionUpperTriangularReal",
  LUDecomposition[#, TargetStructure → "Classic"] &,
  maxMatrix, "UpperTriangularRealMatrix"];

timedReport["DotInteger", #.# &, maxMatrix, "IntegerMatrix"];
timedReport["InverseInteger", Inverse, maxMatrix / 20, "IntegerMatrix"];
timedReport["LinearSolveInteger",
  LinearSolve[#, Sort[#[[All, 1]]] &, maxMatrix / 5, "IntegerMatrix"];
timedReport["MatrixPowerInteger", MatrixPower[#, 5] &, maxMatrix / 5, "IntegerMatrix"];
timedReport["DetInteger", Det, 700, "IntegerMatrix"];
timedReport["TransposeInteger", Transpose, maxMatrix, "IntegerMatrix"];
timedReport["FlattenInteger", Flatten, maxMatrix, "IntegerMatrix"];
timedReport["MeanInteger", Mean, maxVector, "IntegerVector"];
timedReport["SortInteger", Sort, maxVector, "IntegerVector"];
timedReport["SortCustomInteger", SortBy[#, Abs] &, maxVector, "IntegerVector"];
timedReport["EigenvaluesInteger", Eigenvalues, maxMatrix / 30, "IntegerMatrix"];
timedReport["ElementPowerInteger", #^5 &, maxMatrix, "IntegerMatrix"];
timedReport["AddOneInteger", # + 1 &, maxMatrix, "IntegerMatrix"];

timedReport["MatrixExpInteger", MatrixExp, 30, "IntegerMatrix"];

timedReport["MatrixRankInteger", MatrixRank, maxMatrix / 2, "IntegerMatrix"];
timedReport["SingularValuesInteger",
  SingularValueList, maxMatrix / 50, "IntegerMatrix"];
timedReport["QRDecompositionInteger",
  QRDecomposition, maxMatrix / 50, "IntegerMatrix"];

timedReport["DotExtended", #.# &, maxMatrix / 25, "ExtendedMatrix"];
timedReport["InverseExtended", Inverse, maxMatrix / 25, "ExtendedMatrix"];
timedReport["CholeskyExtended",
```

```
  CholeskyDecomposition[Transpose[#].#] &, maxMatrix / 20, "ExtendedMatrix"];
timedReport["EigenvaluesExtended", Eigenvalues, 60, "ExtendedMatrix"];
timedReport["EigenvectorsExtended", Eigenvectors, 80, "ExtendedMatrix"];
timedReport["ElementPowerExtended", #^5 &, maxMatrix / 2, "ExtendedMatrix"];
timedReport["LinearSolveExtended",
  LinearSolve[#, Sort[#[[All, 1]]] &, maxMatrix / 20, "ExtendedMatrix"];
timedReport["DetExtended", Det, maxMatrix / 10, "ExtendedMatrix"];
timedReport["TransposeExtended", Transpose, maxMatrix, "ExtendedMatrix"];
timedReport["FlattenExtended", Flatten, maxMatrix, "ExtendedMatrix"];

timedReport["MeanExtended", Mean, maxVector / 10, "ExtendedVector"];
timedReport["SortExtended", Sort, maxVector, "ExtendedVector"];
timedReport["SortCustomExtended", SortBy[#, Abs] &, 2 * maxVector, "ExtendedVector"];
timedReport["EigenvectorsExtended", Eigenvectors, maxMatrix / 25, "ExtendedMatrix"];
timedReport["MatrixPowerExtended",
  MatrixPower[#, 5] &, maxMatrix / 25, "ExtendedMatrix"];
timedReport["MatrixExpExtended", MatrixExp, 100, "ExtendedMatrix"];


timedReport["MatrixRankExtended", MatrixRank, maxMatrix / 10, "ExtendedMatrix"];
timedReport["LeastSquaresExtended",
  LeastSquares[#, Sort[#[[All, 1]]] &, maxMatrix / 10, "ExtendedMatrix"];
timedReport["LUDecompositionExtended",
  LUDecomposition[#, TargetStructure → "Classic"] &, maxMatrix / 10, "ExtendedMatrix"];
timedReport["QRDecompositionExtended",
  QRDecomposition, maxMatrix / 10, "ExtendedMatrix"];
timedReport["SingularValuesExtended",
  SingularValueList, maxMatrix / 10, "ExtendedMatrix"];
timedReport["AddOneExtended", # + 1 &, maxMatrix / 2, "ExtendedMatrix"];

timedReport["DotExtended1000", #.# &, maxMatrix / 20, "ExtendedMatrix1000"];
timedReport["InverseExtended1000", Inverse, maxMatrix / 25, "ExtendedMatrix1000"];
timedReport["CholeskyExtended1000",
  CholeskyDecomposition[Transpose[#].#] &, maxMatrix / 20, "ExtendedMatrix1000"];
timedReport["EigenvaluesExtended1000", Eigenvalues, 80, "ExtendedMatrix1000"];
timedReport["EigenvectorsExtended1000", Eigenvectors, 80, "ExtendedMatrix1000"];
timedReport["ElementPowerExtended1000", #^5 &, maxMatrix / 2, "ExtendedMatrix1000"];
timedReport["LinearSolveExtended1000",
  LinearSolve[#, Sort[#[[All, 1]]] &, maxMatrix / 25, "ExtendedMatrix1000"];
timedReport["DetExtended1000", Det, maxMatrix / 10, "ExtendedMatrix1000"];
timedReport["TransposeExtended1000", Transpose, maxMatrix, "ExtendedMatrix1000"];
timedReport["FlattenExtended1000", Flatten, maxMatrix, "ExtendedMatrix1000"];

timedReport["MeanExtended1000", Mean, maxVector / 10, "ExtendedVector1000"];
timedReport["SortExtended1000", Sort, maxVector, "ExtendedVector1000"];
timedReport["SortCustomExtended1000",
  SortBy[#, Abs] &, maxVector, "ExtendedVector1000"];
timedReport["MatrixPowerExtended1000",
  MatrixPower[#, 5] &, maxMatrix / 25, "ExtendedMatrix1000"];
```

```
timedReport["MatrixExpExtended1000", MatrixExp, 60, "ExtendedMatrix1000"];

timedReport["MatrixRankExtended1000",
  MatrixRank, maxMatrix / 20, "ExtendedMatrix1000"];
timedReport["LeastSquaresExtended1000",
  LeastSquares[#, Sort[#[[All, 1]]] &, maxMatrix / 10, "ExtendedMatrix1000"];
timedReport["LUDecompositionExtended1000",
  LUDecomposition[#, TargetStructure → "Classic"] &, maxMatrix / 10, "ExtendedMatrix"];
timedReport["SingularValuesExtended1000",
  SingularValueList, 150, "ExtendedMatrix1000"];
timedReport["QRDecompositionExtended1000",
  QRDecomposition, 150, "ExtendedMatrix1000"];
timedReport["AddOneExtended1000", # + 1 &, maxMatrix / 2, "ExtendedMatrix1000"];

evaluateTest["HighPrecision",
  {{"Pi", Hold[N[Pi, 10^6]]}, {"Sqrt", Hold[N[Sqrt[2], 10^6]]},
   {"Sin", Hold[N[Sin[1] + 1 / (1 + Sin[1]), 10^6]]}, {"Exp", Hold[N[Exp[1], 10^6]]},
   {"Log", Hold[N[Log[5], 10^6]]}, {"Tan", Hold[N[Tan[1 + I], 10^6]]},
   {"BesselK", Hold[N[BesselK[2, 3], 5000]]}, {"Gamma", Hold[N[Gamma[11 / 3], 5000]]},
   {"GoldenRatio", Hold[N[GoldenRatio, 10^6]]}, {"Catalan", Hold[N[Catalan, 10^6]]},
   {"EulerGamma", Hold[N[EulerGamma, 10^6]]}, {"PlasticConstant",
    Hold[N[Root[x^3 - x - 1, 1], 10^6]]}, {"Glaisher", Hold[N[Glaisher, 5000]]}}];

evaluateTest["RandomNumbers", {{"RandomReal", Hold[makeData["RealMatrix", 2000]]},
   {"RandomInteger", Hold[makeData["IntegerMatrix", 2000]]},
   {"RandomExtended", Hold[makeData["ExtendedMatrix", 2000]]},
   {"RandomExtended1000", Hold[makeData["ExtendedMatrix1000", 2000]]},
   {"RandomNormal", Hold[RandomVariate[NormalDistribution[0, 1], 10^7]]},
   {"RandomPoisson", Hold[RandomVariate[PoissonDistribution[4], 10^7]]},
   {"RandomBinomial", Hold[RandomVariate[BinomialDistribution[10, 0.2], 10^7]]},
   {"RandomExponential", Hold[RandomVariate[ExponentialDistribution[2], 10^7]]},
   {"RandomHypergeometric",
    Hold[RandomVariate[HypergeometricDistribution[50, 50, 100], 10^7]]},
   {"RandomGeometric", Hold[RandomVariate[GeometricDistribution[1 / 3], 10^7]]}
  }];

timedReport["FourierRealGPU", Fourier, 10 * maxVector, "RealVectorGPU"];
timedReport["SortRealGPU", Sort, maxVector, "RealVectorGPU"];
timedReport["MeanRealGPU", Mean, 1000 maxVector, "RealVectorGPU"];
timedReport["DotRealGPU", #.# &, 2 * maxMatrix, "RealMatrixGPU"];
timedReport["InverseRealGPU", Inverse, maxMatrix, "RealMatrixGPU"];
Quiet@timedReport["LinearSolveRealGPU",
   LinearSolve[#, Sort[#[[All, 1]]] &, 2 * maxMatrix, "RealMatrixGPU"];
timedReport["CholeskyRealGPU",
  CholeskyDecomposition[Transpose[#].#] &, maxMatrix, "RealMatrixGPU"];
timedReport["MatrixPowerRealGPU", MatrixPower[#, 5] &, maxMatrix, "RealMatrixGPU"];
timedReport["DetRealGPU", Det, 2 * maxMatrix, "RealMatrixGPU"];
timedReport["TransposeRealGPU", Transpose, maxMatrix, "RealMatrixGPU"];
timedReport["FlattenRealGPU", Flatten, maxMatrix, "RealMatrixGPU"];
```

```
timedReport["EigenvaluesRealGPU", Eigenvalues, maxMatrix / 2, "RealMatrixGPU"];
timedReport["EigenvectorsRealGPU", Eigenvectors, maxMatrix / 2, "RealMatrixGPU"];
timedReport["ElementPowerRealGPU", #^5 &, maxMatrix, "RealMatrixGPU"];
timedReport["MovingAverageRealGPU",
  MovingAverage[#, 10] &, maxVector, "RealVectorGPU"];
timedReport["FitRealGPU", (Function[{data}, Fit[data, x, x]][Transpose[{#, #}]]) &,
  4 * maxVector, "RealVectorGPU"];
timedReport["MatrixExpRealGPU", MatrixExp, maxMatrix / 5, "RealMatrixGPU"];
timedReport["MatrixRankRealGPU", MatrixRank, 2 * maxMatrix, "RealMatrixGPU"];
timedReport["LeastSquaresRealGPU",
  LeastSquares[#, Sort[#[[All, 1]]] &, 2 * maxMatrix, "RealMatrixGPU"];
timedReport["LUDecompositionReal",
  LUDecomposition[#, TargetStructure → "Classic"] &, 2 * maxMatrix, "RealMatrix"];
timedReport["SingularValuesRealGPU",
  SingularValueList, 2 * maxMatrix, "RealMatrixGPU"];
timedReport["QRDecompositionRealGPU", QRDecomposition, maxMatrix, "RealMatrixGPU"];
timedReport["AddOneRealGPU", # + 1 &, 2 * maxMatrix, "RealMatrixGPU"];

timedReport["FourierComplexGPU", Fourier, 4 * maxVector, "ComplexVectorGPU"];
timedReport["SortCustomComplexGPU", SortBy[#, Abs] &, maxVector, "ComplexVectorGPU"];
timedReport["MeanComplexGPU", Mean, 10 maxVector, "ComplexVectorGPU"];
timedReport["DotComplexGPU", #.# &, maxMatrix, "ComplexMatrixGPU"];
timedReport["InverseComplexGPU", Inverse, maxMatrix, "ComplexMatrixGPU"];
Quiet@timedReport["LinearSolveComplexGPU",
   LinearSolve[#, Sort[#[[All, 1]]] &, maxMatrix, "ComplexMatrixGPU"];
timedReport["MatrixPowerComplexGPU",
  MatrixPower[#, 5] &, maxMatrix, "ComplexMatrixGPU"];
timedReport["DetComplexGPU", Det, maxMatrix, "ComplexMatrixGPU"];
timedReport["TransposeComplexGPU", Transpose, maxMatrix, "ComplexMatrixGPU"];
timedReport["FlattenComplexGPU", Flatten, maxMatrix, "ComplexMatrixGPU"];
timedReport["EigenvaluesComplexGPU", Eigenvalues, maxMatrix, "ComplexMatrixGPU"];
timedReport["EigenvectorsComplexGPU", Eigenvectors, maxMatrix, "ComplexMatrixGPU"];
timedReport["ElementPowerComplexGPU", #^5 &, maxMatrix, "ComplexMatrixGPU"];
timedReport["LeastSquaresComplexGPU",
  LeastSquares[#, Sort[#[[All, 1]]] &, maxMatrix, "ComplexMatrixGPU"];
timedReport["LUDecompositionComplexGPU",
  LUDecomposition[#, TargetStructure → "Classic"] &, maxMatrix, "ComplexMatrix"];
timedReport["QRDecompositionComplexGPU",
  QRDecomposition, maxMatrix, "ComplexMatrixGPU"];
timedReport["SingularValuesComplexGPU",
  SingularValueList, maxMatrix, "ComplexMatrixGPU"];
timedReport["MovingAverageComplexGPU",
  MovingAverage[#, 10] &, maxVector, "ComplexVectorGPU"];
timedReport["FitComplexGPU", (Function[{data}, Fit[data, x, x]][Transpose[{#, #}]]) &,
  4 * maxVector, "ComplexVectorGPU"];
timedReport["MatrixExpComplexGPU", MatrixExp, maxMatrix / 5, "ComplexMatrixGPU"];
timedReport["MatrixRankComplexGPU", MatrixRank, 3 / 2 maxMatrix, "ComplexMatrixGPU"];
timedReport["AddOneComplexGPU", # + 1 &, maxMatrix, "ComplexMatrixGPU"];
```

```
timedReport["SortIntegerGPU", Sort, maxVector, "IntegerVectorGPU"];
timedReport["MeanIntegerGPU", Mean, 10 maxVector, "IntegerVectorGPU"];
timedReport["DotIntegerGPU", #.# &, maxMatrix, "IntegerMatrixGPU"];
Quiet@timedReport["LinearSolveIntegerGPU",
   LinearSolve[#, Sort[#[[All, 1]]] &, 500, "IntegerMatrixGPU"];
timedReport["MatrixPowerIntegerGPU", MatrixPower[#, 5] &, 500, "IntegerMatrixGPU"];
timedReport["MatrixExpIntegerGPU", MatrixExp, 30, "IntegerMatrixGPU"];
timedReport["DetIntegerGPU", Det, 700, "IntegerMatrixGPU"];
timedReport["TransposeIntegerGPU", Transpose, maxMatrix, "IntegerMatrixGPU"];
timedReport["FlattenIntegerGPU", Flatten, maxMatrix, "IntegerMatrixGPU"];
timedReport["EigenvaluesIntegerGPU", Eigenvalues, 70, "IntegerMatrixGPU"];
timedReport["ElementPowerIntegerGPU", #^5 &, maxMatrix, "IntegerMatrixGPU"];
timedReport["MatrixRankIntegerGPU", MatrixRank, maxMatrix / 2, "IntegerMatrixGPU"];
timedReport["SingularValuesIntegerGPU",
  SingularValueList, maxMatrix / 50, "IntegerMatrixGPU"];
timedReport["QRDecompositionIntegerGPU",
  QRDecomposition, maxMatrix / 50, "IntegerMatrixGPU"];
timedReport["AddOneIntegerGPU", # + 1 &, maxMatrix, "IntegerMatrixGPU"];

timedReport["FourierComplexGPU", Fourier, 5 * maxVector, "ComplexVectorGPU"];
timedReport["MeanComplexGPU", Mean, 10 maxVector, "ComplexVectorGPU"];
timedReport["DotCompleGPU", #.# &, maxMatrix, "ComplexMatrixGPU"];
timedReport["InverseComplexGPU", Inverse, maxMatrix, "ComplexMatrixGPU"];
timedReport["LinearSolveComplexGPU",
  LinearSolve[#, Sort[#[[All, 1]]] &, maxMatrix, "ComplexMatrixGPU"];
timedReport["MatrixPowerComplexGPU",
  MatrixPower[#, 5] &, maxMatrix / 2, "ComplexMatrixGPU"];
timedReport["DetComplexGPU", Det, maxMatrix, "ComplexMatrixGPU"];
timedReport["EigenvaluesComplexGPU",
  Eigenvalues, maxMatrix / 2, "ComplexMatrixGPU"];
timedReport["EigenvectorsComplexGPU",
  Eigenvectors, maxMatrix / 2, "ComplexMatrixGPU"];
timedReport["TransposeComplexGPU", Transpose, maxMatrix, "ComplexMatrixGPU"];
timedReport["FlattenComplexGPU", Flatten, maxMatrix, "ComplexMatrixGPU"];
timedReport["ElementPowerComplexGPU", #^5 &, maxMatrix, "ComplexMatrixGPU"];
timedReport["MatrixExpComplexGPU", MatrixExp, maxMatrix / 2, "ComplexMatrixGPU"];
timedReport["MatrixRankComplexGPU", MatrixRank, 3 / 2 maxMatrix, "ComplexMatrixGPU"];


timedReport["RowReduceReal", RowReduce, maxMatrix, "RealMatrix"];
timedReport["RowReduceComplex", RowReduce, maxMatrix, "ComplexMatrix"];
timedReport["RowReduceInteger", RowReduce, maxMatrix, "IntegerMatrix"];
timedReport["RowReduceExtended", RowReduce, maxMatrix / 10, "ExtendedMatrix"];
timedReport["RowReduceExtended1000", RowReduce, maxMatrix / 10, "ExtendedMatrix1000"];
timedReport["RowReduceRealGPU", RowReduce, maxMatrix, "RealMatrixGPU"];
timedReport["RowReduceComplexGPU", RowReduce, maxMatrix, "ComplexMatrixGPU"];
timedReport["RowReduceIntegerGPU", RowReduce, maxMatrix, "IntegerMatrixGPU"];

timedReport["NullSpaceReal", NullSpace, maxMatrix, "RealMatrix"];
```

```
timedReport["NullSpaceComplex", NullSpace, maxMatrix, "ComplexMatrix"];
timedReport["NullSpaceInteger", NullSpace, maxMatrix, "IntegerMatrix"];
timedReport["NullSpaceExtended", NullSpace, maxMatrix / 20, "ExtendedMatrix"];
timedReport["NullSpaceExtended1000", NullSpace, maxMatrix / 20, "ExtendedMatrix1000"];
timedReport["NullSpaceRealGPU", NullSpace, maxMatrix, "RealMatrixGPU"];
timedReport["NullSpaceComplexGPU", NullSpace, maxMatrix, "ComplexMatrixGPU"];
timedReport["NullSpaceIntegerGPU", NullSpace, maxMatrix, "IntegerMatrixGPU"];

timedReport["ScalarMultiplyReal", # * 3 &, maxMatrix, "RealMatrix"];
timedReport["ScalarMultiplyComplex", # * 3 &, maxMatrix, "ComplexMatrix"];
timedReport["ScalarMultiplyInteger", # * 3 &, maxMatrix, "IntegerMatrix"];
timedReport["ScalarMultiplyExtended", # * 3 &, maxMatrix, "ExtendedMatrix"];
timedReport["ScalarMultiplyExtended1000", # * 3 &, maxMatrix, "ExtendedMatrix1000"];
timedReport["ScalarMultiplyRealGPU", # * 3 &, maxMatrix, "RealMatrixGPU"];
timedReport["ScalarMultiplyComplexGPU", # * 3 &, maxMatrix, "ComplexMatrixGPU"];
timedReport["ScalarMultiplyIntegerGPU", # * 3 &, maxMatrix, "IntegerMatrixGPU"];

timedReport["DeleteRowReal", Delete[#, 3] &, maxMatrix, "RealMatrix"];
timedReport["DeleteRowComplex", Delete[#, 3] &, maxMatrix, "ComplexMatrix"];
timedReport["DeleteRowInteger", Delete[#, 3] &, maxMatrix, "IntegerMatrix"];
timedReport["DeleteRowExtended", Delete[#, 3] &, maxMatrix, "ExtendedMatrix"];
timedReport["DeleteRowExtended1000", Delete[#, 3] &, maxMatrix, "ExtendedMatrix1000"];
timedReport["DeleteRowRealGPU", Delete[#, 3] &, maxMatrix, "RealMatrixGPU"];
timedReport["DeleteRowComplexGPU", Delete[#, 3] &, maxMatrix, "ComplexMatrixGPU"];
timedReport["DeleteRowIntegerGPU", Delete[#, 3] &, maxMatrix, "IntegerMatrixGPU"];

timedReport["DeleteColumnReal", DeleteColumns[#, {3}] &, maxMatrix, "RealMatrix"];
timedReport["DeleteColumnComplex",
  DeleteColumns[#, {3}] &, maxMatrix, "ComplexMatrix"];
timedReport["DeleteColumnInteger",
  DeleteColumns[#, {3}] &, maxMatrix, "IntegerMatrix"];
timedReport["DeleteColumnExtended",
  DeleteColumns[#, {3}] &, maxMatrix, "ExtendedMatrix"];
timedReport["DeleteColumnExtended1000",
  DeleteColumns[#, {3}] &, maxMatrix, "ExtendedMatrix1000"];
timedReport["DeleteColumnRealGPU",
  DeleteColumns[#, {3}] &, maxMatrix, "RealMatrixGPU"];
timedReport["DeleteColumnComplexGPU",
  DeleteColumns[#, {3}] &, maxMatrix, "ComplexMatrixGPU"];
timedReport["DeleteColumnIntegerGPU",
  DeleteColumns[#, {3}] &, maxMatrix, "IntegerMatrixGPU"];
```

Maple

```
mapleVersion := “Maple2026”:
with(LinearAlgebra):
with(DiscreteTransforms):
with(combinat, fibonacci):
with(Statistics):
with(Student[Statistics]):
```

```
with(RandomTools[MersenneTwister]):
with(Optimization):
with(RandomTools):
with(stats):
with(combinat):
with(ArrayTools):
currentdir("/Users/test/Desktop/BenchmarkData"):
steps := 6:
repeats := 4:
maxVector := 10^5:
maxMatrix := 2000:
maxSparseVector:= 2 * 10^6:
maxSparseMatrix := 20000:

RandomSparseVector := proc(n::posint)
  local v, k;
  v := Vector(n, storage = sparse, fill = 1.);
  for k to floor(n/10000) do
    v[rand(1..n)()] := Generate(float);
  end do;
  return v;
end proc:

RandomSparseMatrix := proc(m::posint)
  local A, k;
  A := Matrix(m, m, storage = sparse);
  for k to floor( m * m / 10000) do
    A[rand(1..m)(), rand(1..m)()] := Generate(float);
  end do;
  return A;
end proc:

makeData := proc(type, size)
 local dat, i;
 if type = "Vector" then dat := RandomVector[row](size, generator = 0. .. 1.0);
elif type = "Matrix" then dat := Generate('Matrix'(float(range = -10 .. 10, method = uniform), size, size));
elif type = "VectorFloat8" then dat := RandomVector[row](size, generator = 0. .. 1.0, datatype = float[8]);
elif type = "MatrixFloat8" then dat := RandomMatrix(size, size, generator = -10.0 .. 10.0, datatype = float[8]);
elif type = "UpperTriangularFloat8" then dat := UpperTriangle(1.0+makeData("MatrixFloat8", size));
elif type = "UpperTriangularMatrix" then dat := UpperTriangle(1.0+makeData("Matrix", size));
elif type = "SparseMatrix" then dat := RandomSparseMatrix(size);
elif type = "SparseVector" then dat := RandomSparseVector(size);
elif type = "IntegerMatrix" then dat := Generate('Matrix'(integer(range = -10 .. 10), size, size));
elif type = "IntegerVector" then dat := Generate('Vector'(integer(range = -10 .. 10), size));
elif type = "IntegerMatrix8" then dat := Generate('Matrix'(integer(range = -10 .. 10), size, size, datatype = integer[8]));
elif type = "IntegerVector8" then dat := Generate('Vector'(integer(range = -10 .. 10), size, datatype = integer[8]));
elif type = "BigNumberMatrix" then dat := RandomMatrix(size, size, generator =(proc(x) 20*GenerateFloat(digits = 50) - 10; end proc));
elif type = "BigNumberVector" then dat := RandomVector[row](size, generator =(proc(x) 20*GenerateFloat(digits = 50) - 10; end proc));
elif type = "BigNumberMatrix1000" then dat := RandomMatrix(size, size, generator =(proc(x) 20*GenerateFloat(digits = 1000) - 10; end proc));
elif type = "BigNumberVector1000" then dat := RandomVector[row](size, generator =(proc(x) 20*GenerateFloat(digits = 1000) - 10; end proc));
elif type = "ComplexVector" then dat := makeData("Vector", size) + makeData("Vector", size)*I;
elif type = "ComplexMatrix" then dat := makeData("Matrix", size) + makeData("Matrix", size)*I;
elif type = "ComplexVector8" then dat := makeData("VectorFloat8", size) + makeData("VectorFloat8", size)*I;
elif type = "ComplexMatrix8" then dat := makeData("MatrixFloat8", size) + makeData("MatrixFloat8", size)*I;
else print(type); end if; dat; end proc:

timedDataOperation := proc(expr, size, type)
 local totaltime, data, i;
 totaltime := 0;
 for i to repeats do
```

```
    data := makeData(type, round(size));
    totaltime := totaltime + time[real](expr(data));
  end do;
  totaltime/repeats;
end proc:

timedReport := proc(filename, expr, hi, type)
  local s;
  print(filename);
  ExportMatrix(cat(mapleVersion, filename, ".txt"), convert([seq([floor(s), timedDataOperation('expr', floor(s), type)], s = hi/steps .. hi, hi/steps)], Matrix));
end proc:

numericfntest := proc(fnrow, n, type)
  local data, label, fn, scale;
  label := fnrow[1];
  fn := fnrow[2];
  scale := fnrow[3];
  data := makeData(type, round(n*scale));
  [label, time[real](map(fn, data))];
end proc:

numerictest := proc(file, fns, n, type)
  local i;
  ExportMatrix(cat(mapleVersion, file, ".txt"), Matrix([seq(numericfntest([fns[i, 1], fns[i, 2], fns[i, 3]], n, type), i = 1 .. LinearAlgebra:-RowDimension(fns))]));
end proc:

numerictestinplace := proc(file, fns, n, type)
  local i;
  ExportMatrix(cat(mapleVersion, file, ".txt"), Matrix([seq(numericfntestinplace([fns[i, 1], fns[i, 2], fns[i, 3]], n, type), i = 1 .. LinearAlgebra:-RowDimen-
sion(fns))]));
end proc:

numerictestelementwise := proc(file, fns, n, type)
  local i;
  ExportMatrix(cat(mapleVersion, file, ".txt"), Matrix([seq(elementwisefntest([fns[i, 1], fns[i, 2], fns[i, 3]], n, type), i = 1 .. LinearAlgebra:-RowDimen-
sion(fns))]));
end proc:

numericfntestinplace := proc(fnrow, n, type)
  local data, label, fn, scale;
  label := fnrow[1];
  fn := fnrow[2];
  scale := fnrow[3];
  data := makeData(type, round(n*scale));
  [label, time[real](Map(fn, data))];
end proc:

elementwisefntest := proc(fnrow, n, type)
  local data, label, fn, scale;
  label := fnrow[1];
  fn := fnrow[2];
  scale := fnrow[3];
  data := makeData(type, round(n*scale));
  [label, time[real](fn(data))];
end proc:

highPrecisionTest := proc(file, fns, n)
  local i;
  ExportMatrix(cat(mapleVersion, file, ".txt"), Matrix([seq(time[real](evalf(i, n)), i in fns)]));
end proc:
```

```
evaluateTest := proc(file, rows)
 local i, n, output;
 n := nops(rows);
 output := Matrix(n, 2);
 for i to n do
  output[i, 1] := rows[i][1];
  output[i, 2] := time[real](eval(rows[i][2]));
 end do;
 ExportMatrix(cat(mapleVersion, file, ".txt"), output);
end proc:

RandomInteger := proc(x, y) RandomTools[Generate](integer(range = 1 .. 100)); end proc:

fnList := Matrix([
 ["sqrt", sqrt, 1], ["sin", sin, 1], ["cos", cos, 1], ["tan", tan, 1],
 ["arcsin", arcsin, 1], ["arccos", arccos, 1], ["arctan", arctan, 1],
 ["sec", sec, 1], ["csc", csc, 1], ["cot", cot, 1], ["exp", exp, 1],
 ["sinh", sinh, 1], ["cosh", cosh, 1], ["tanh", tanh, 1], ["log", log, 1],
 ["log10", log10, 1], ["erf", erf, 1],

 ["GAMMA", GAMMA, 0.5],
 ["BesselJ0", proc(x) BesselJ(0, x); end proc, 0.5],
 ["BesselK1", proc(x) BesselK(1, x); end proc, 0.5],
 ["BesselY3", proc(x) BesselY(3, x); end proc, 0.5],

 ["arcsinh", arcsinh, 1], ["arccosh1p", proc(x) arccosh(x + 1); end proc, 1], ["arctanh", arctanh, 1],
 ["Zeta", Zeta, 1], ["composite1", proc(x) x*sin(log(x) + 2*sqrt(x) + 3) + sqrt(x) + log(x); end proc, 1],
 ["sin_cos", proc(x) sin(cos(x)); end proc, 1], ["log_sinh", proc(x) log(sinh(x)); end proc, 1],
 ["erf_sin", proc(x) erf(sin(x)); end proc, 1], ["exp_x2", proc(x) exp(x^2); end proc, 1], ["Psi", Psi, 1],
 ["x_plus_1", proc(x) x + 1; end proc, 1], ["inv_x", proc(x) 1/x; end proc, 1], ["x2", proc(x) x^2; end proc, 1],
 ["frem_0p5", proc(x) frem(x, 0.5); end proc, 1]
]):
elementwisefnList := Matrix([
 ["sqrt", `~`[sqrt], 1], ["sin", `~`[sin], 1], ["cos", `~`[cos], 1], ["tan", `~`[tan], 1],
 ["arcsin", `~`[arcsin], 1], ["arccos", `~`[arccos], 1], ["arctan", `~`[arctan], 1],
 ["sec", `~`[sec], 1], ["csc", `~`[csc], 1], ["cot", `~`[cot], 1], ["exp", `~`[exp], 1],
 ["sinh", `~`[sinh], 1],["cosh", `~`[cosh], 1], ["tanh", `~`[tanh], 1], ["log", `~`[log], 1],
 ["log10", `~`[log10], 1], ["erf", `~`[erf], 1],

 ["GAMMA", `~`[GAMMA], 0.5],
 ["BesselJ0", `~`[proc(x) BesselJ(0, x); end proc], 0.5],
 ["BesselK1", `~`[proc(x) BesselK(1, x); end proc], 0.5],
 ["BesselY3", `~`[proc(x) BesselY(3, x); end proc], 0.5],

 ["arcsinh", `~`[arcsinh], 1], ["arccosh1p", `~`[proc(x) arccosh(x + 1); end proc], 1], ["arctanh", `~`[arctanh], 1],
 ["Zeta", `~`[Zeta], 1],["composite1", `~`[proc(x) x*sin(log(x) + 2*sqrt(x) + 3) + sqrt(x) + log(x); end proc], 1],
 ["sin_cos", `~`[proc(x) sin(cos(x)); end proc], 1], ["log_sinh", `~`[proc(x) log(sinh(x)); end proc], 1],
 ["erf_sin", `~`[proc(x) erf(sin(x)); end proc], 1], ["exp_x2", `~`[proc(x) exp(x^2); end proc], 1],["Psi", `~`[Psi], 1],
 ["x_plus_1", `~`[proc(x) x + 1; end proc], 1], ["inv_x", `~`[proc(x) 1/x; end proc], 1], ["x2", `~`[proc(x) x^2; end proc], 1],
 ["frem_0p5", `~`[proc(x) frem(x, 0.5); end proc], 1]
]):

fnListZeroSafe:=Matrix([
 ["sqrt", sqrt, 1], ["sin", sin, 1], ["cos", cos, 1], ["tan", tan, 1],
 ["arcsin", arcsin, 1], ["arccos", arccos, 1], ["arctan", arctan, 1],
 ["sec", sec, 1], ["csc", proc(x) csc(x+1); end proc, 1], ["cot", proc(x) cot(x+1); end proc, 1], ["exp", exp, 1],
 ["sinh", sinh, 1], ["cosh", cosh, 1], ["tanh", tanh, 1], ["log", proc(x) log(x+1); end proc, 1],
 ["log10", proc(x) log10(1+x); end proc, 1], ["erf", erf, 1],
```

```
  ["GAMMA", proc(x) GAMMA(x+1); end proc, 0.5],
  ["BesselJ0", proc(x) BesselJ(0, x); end proc, 0.5],
  ["BesselK1", proc(x) BesselK(1, x+1); end proc, 0.5],
  ["BesselY3", proc(x) BesselY(3, x+1); end proc, 0.5],

  ["arcsinh", arcsinh, 1], ["arccosh1p", proc(x) arccosh(x + 1); end proc, 1], ["arctanh", arctanh, 1],
  ["Zeta", Zeta, 1], ["composite1", proc(x) x*sin(log(x+1) + 2*sqrt(x) + 3) + sqrt(x) + log(x+1); end proc, 1],
  ["sin_cos", proc(x) sin(cos(x)); end proc, 1], ["log_sinh", proc(x) log(sinh(x+1)); end proc, 1],
  ["erf_sin", proc(x) erf(sin(x)); end proc, 1], ["exp_x2", proc(x) exp(x^2); end proc, 1], ["Psi", proc(x) Psi(x+1) end proc, 1],
  ["x_plus_1", proc(x) x + 1; end proc, 1], ["inv_x", proc(x) 1/(x+1); end proc, 1], ["x2", proc(x) x^2; end proc, 1],
  ["frem_0p5", proc(x) frem(x, 0.5); end proc, 1]
]):

elementwisefnListZeroSafe := Matrix([
  ["sqrt", `~`[sqrt], 1], ["sin", `~`[sin], 1], ["cos", `~`[cos], 1], ["tan", `~`[tan], 1],
  ["arcsin", `~`[arcsin], 1], ["arccos", `~`[arccos], 1], ["arctan", `~`[arctan], 1],
  ["sec", `~`[sec], 1], ["csc", `~`[proc(x) csc(x+1); end proc], 1], ["cot", `~`[proc(x) cot(x+1); end proc], 1], ["exp", `~`[exp], 1],
  ["sinh", `~`[sinh], 1],["cosh", `~`[cosh], 1], ["tanh", `~`[tanh], 1], ["log", `~`[proc(x) log(x+1); end proc], 1],
  ["log10", `~`[proc(x) log10(1+x); end proc], 1], ["erf", `~`[erf], 1],

  ["GAMMA", `~`[proc(x) GAMMA(x+1); end proc], 0.5],
  ["BesselJ0", `~`[proc(x) BesselJ(0, x); end proc], 0.5],
  ["BesselK1", `~`[proc(x) BesselK(1, x+1); end proc], 0.5],
  ["BesselY3", `~`[proc(x) BesselY(3, x+1); end proc], 0.5],

  ["arcsinh", `~`[arcsinh], 1], ["arccosh1p", `~`[proc(x) arccosh(x + 1); end proc], 1], ["arctanh", `~`[arctanh], 1],
  ["Zeta", `~`[Zeta], 1],["composite1", `~`[proc(x) x*sin(log(x+1) + 2*sqrt(x) + 3) + sqrt(x) + log(x+1); end proc], 1],
  ["sin_cos", `~`[proc(x) sin(cos(x)); end proc], 1], ["log_sinh", `~`[proc(x) log(sinh(x+1)); end proc], 1],
  ["erf_sin", `~`[proc(x) erf(sin(x)); end proc], 1], ["exp_x2", `~`[proc(x) exp(x^2); end proc], 1],["Psi", `~`[proc(x) Psi(x+1) end proc], 1],
  ["x_plus_1", `~`[proc(x) x + 1; end proc], 1], ["inv_x", `~`[proc(x) 1/(x+1); end proc], 1], ["x2", `~`[proc(x) x^2; end proc], 1],
  ["frem_0p5", `~`[proc(x) frem(x, 0.5); end proc], 1]
]):

steps := 3:
repeats := 3:
timedReport("MatrixExpInteger8", 'MatrixExponential', 3, "IntegerMatrix8");
timedReport("MatrixExpInteger", 'MatrixExponential', 3, "IntegerMatrix");
repeats := 5:
steps := 6:

timedReport("DotSparse", 'proc(x) x . x; end proc', maxSparseMatrix, "SparseMatrix");
timedReport("FlattenSparse", 'proc(m) convert(m, Vector[row]); end proc', maxSparseMatrix / 10, "SparseMatrix");
timedReport("MatrixPowerSparse", 'proc(x) x^5; end proc', 2 * maxSparseMatrix / 5, "SparseMatrix");
timedReport("ElementPowerSparse", 'proc(m) map(proc(x) x^5; end proc, m); end proc', maxSparseMatrix / 4, "SparseMatrix");
timedReport("MeanSparse", 'Statistics:-Mean', 100*maxVector, "SparseVector");
timedReport("LinearSolveSparse", 'proc(x) LinearSolve(x, sort(Column(x, 1)), methodoptions = [maxiter = 10000]); end proc', maxSparseMatrix,
"SparseMatrix");
timedReport("TransposeSparse", 'Transpose', 5 * maxSparseMatrix, "SparseMatrix");
timedReport("AddOneSparse", 'proc(m) m+~1 end proc', maxSparseMatrix, "SparseMatrix");
timedReport("FourierReal", 'FourierTransform', 10* maxVector, "Vector");
timedReport("SortReal", 'sort', maxVector, "Vector");
timedReport("MeanReal", 'Statistics:-Mean', 10 * maxVector, "Vector");
timedReport("DotReal", 'proc(x) x . x; end proc', 2 *  maxMatrix, "Matrix");
timedReport("InverseReal", 'MatrixInverse', 2 * maxMatrix, "Matrix");
timedReport("LinearSolveReal", 'proc(x) LinearSolve(x, sort(Column(x, 1))); end proc', 2 * maxMatrix, "Matrix");
timedReport("CholeskyReal", 'proc(S) LUDecomposition((Transpose(S)) . S, method = 'Cholesky'); end proc', maxMatrix, "Matrix");
timedReport("MatrixPowerReal", 'proc(x) x^5; end proc', maxMatrix, "Matrix");
timedReport("DetReal", 'Determinant', 2 * maxMatrix, "Matrix");
timedReport("TransposeReal", 'Transpose', maxMatrix, "Matrix");
timedReport("FlattenReal", 'proc(m) convert(m, Vector[row]); end proc', maxMatrix, "Matrix");
```

```
timedReport("EigenvaluesReal", 'Eigenvalues', maxMatrix, "Matrix");
timedReport("EigenvectorsReal", 'Eigenvectors', maxMatrix, "Matrix");
timedReport("LinearProgrammingReal", 'proc(x) LPSolve(abs(Row(x, 1)), [-abs(x), -abs(Row(x, 2))], [0, infinity]); end proc', maxMatrix/4, "Matrix");
timedReport("ElementPowerReal", 'proc(m) map(proc(x) x^5; end proc, m); end proc', maxMatrix, "Matrix");
timedReport("FitReal", 'proc(m) Fit(b*x + a, m, m, x); end proc', 4 * maxVector, "Vector");
timedReport("MatrixExpReal", 'MatrixExponential', maxMatrix/5, "Matrix");
timedReport("CovarianceReal", 'CovarianceMatrix', maxMatrix, "Matrix");
timedReport("MatrixRankReal", 'LinearAlgebra:-Rank', 3*maxMatrix, "Matrix");
timedReport("LUDecompositionReal", 'LUDecomposition', 2 * maxMatrix, "Matrix");
timedReport("LeastSquaresReal", 'proc(x) LeastSquares(x, sort(Column(x, 1))); end proc', 2 * maxMatrix, "Matrix");
timedReport("AddOneReal", 'proc(m) m+~1 end proc', maxMatrix, "Matrix");
timedReport("AddOneFloat8", 'proc(m) m+~1 end proc', 2 * maxMatrix, "MatrixFloat8");
timedReport("FourierFloat8", 'FourierTransform', 10 * maxVector, "VectorFloat8");
timedReport("SortFloat8", 'sort', maxVector, "VectorFloat8");
timedReport("MeanFloat8", 'Statistics:-Mean', 1000 * maxVector, "VectorFloat8");
timedReport("DotFloat8", 'proc(x) x . x; end proc', 2* maxMatrix, "MatrixFloat8");
timedReport("InverseFloat8", 'MatrixInverse', maxMatrix, "MatrixFloat8");
timedReport("LinearSolveFloat8", 'proc(x) LinearSolve(x, sort(Column(x, 1))); end proc', 2 * maxMatrix, "MatrixFloat8");
timedReport("CholeskyFloat8", 'proc(S) LUDecomposition((Transpose(S)) . S, method = 'Cholesky'); end proc', maxMatrix, "MatrixFloat8");
timedReport("MatrixPowerFloat8", 'proc(x) x^5; end proc', maxMatrix, "MatrixFloat8");
timedReport("DetFloat8", 'Determinant', 2 * maxMatrix, "MatrixFloat8");
timedReport("TransposeFloat8", 'Transpose', maxMatrix, "MatrixFloat8");
timedReport("FlattenFloat8", 'proc(m) convert(m, Vector[row]); end proc', maxMatrix, "MatrixFloat8");
timedReport("EigenvaluesFloat8", 'Eigenvalues', 1/2*maxMatrix, "MatrixFloat8");
timedReport("EigenvectorsFloat8", 'Eigenvectors', 1/2*maxMatrix, "MatrixFloat8");
timedReport("LinearProgrammingFloat8", 'proc(x) LPSolve(abs(Row(x, 1)), [-abs(x), -abs(Row(x, 2))], [0, infinity]); end proc',
1/4*maxMatrix, "MatrixFloat8");
timedReport("ElementPowerFloat8", 'proc(m) map(proc(x) x^5; end proc, m); end proc', 1/2*maxMatrix, "MatrixFloat8");
timedReport("FitFloat8", 'proc(m) Fit(b*x + a, m, m, x); end proc', 4 *maxVector, "VectorFloat8");
timedReport("MatrixExpFloat8", 'MatrixExponential', maxMatrix/5, "MatrixFloat8");
timedReport("CovarianceFloat8", 'CovarianceMatrix', maxMatrix, "MatrixFloat8");
timedReport("MatrixRankFloat8", 'LinearAlgebra:-Rank', 2 * maxMatrix, "MatrixFloat8");
timedReport("LUDecompositionFloat8", 'LUDecomposition', maxMatrix, "MatrixFloat8");
timedReport("LeastSquaresFloat8", 'proc(x) LeastSquares(x, sort(Column(x, 1))); end proc', maxMatrix, "MatrixFloat8");
timedReport("LinearProgrammingReal", 'proc(x) LPSolve(abs(Row(x, 1)), [-abs(x), -abs(Row(x, 2))], [0, infinity]); end proc', maxMatrix, "Matrix");
timedReport("LinearProgrammingFloat8", 'proc(x) LPSolve(abs(Row(x, 1)), [-abs(x), -abs(Row(x, 2))], [0, infinity]); end proc', maxMatrix,"MatrixFloat8");
timedReport("FourierComplex", 'FourierTransform', 4 * maxVector, "ComplexVector");
timedReport("DetUpperTriangularReal", 'Determinant', maxMatrix, "UpperTriangularMatrix");
timedReport("DotUpperTriangularReal", 'proc(x) x . x; end proc', maxMatrix, "UpperTriangularMatrix");
timedReport("EigensystemUpperTriangularReal", 'proc(x) [Eigenvalues(x), Eigenvectors(x)]; end proc', maxMatrix, "UpperTriangularMatrix");
timedReport("EigenvaluesUpperTriangularReal", 'Eigenvalues', maxMatrix, "UpperTriangularMatrix");
timedReport("EigenvectorsUpperTriangularReal", 'Eigenvectors', maxMatrix, "UpperTriangularMatrix");
timedReport("InverseUpperTriangularReal", 'MatrixInverse', maxMatrix, "UpperTriangularMatrix");
timedReport("LeastSquaresUpperTriangularReal", 'proc(x) LeastSquares(x, sort(Column(x, 1))); end proc', maxMatrix, "UpperTriangularMatrix");
timedReport("LUDecompositionUpperTriangularReal", 'LUDecomposition', maxMatrix, "UpperTriangularMatrix");
timedReport("MatrixPowerUpperTriangularReal", 'proc(x) x^5; end proc', maxMatrix, "UpperTriangularMatrix");
timedReport("QRDecompositionUpperTriangularReal", 'QRDecomposition', maxMatrix, "UpperTriangularMatrix");
timedReport("DotComplex", 'proc(x) x . x; end proc', maxMatrix, "ComplexMatrix");
timedReport("InverseComplex", 'MatrixInverse', maxMatrix, "ComplexMatrix");
timedReport("LinearSolveComplex", 'proc(x) LinearSolve(x, sort(Column(x, 1))); end proc', maxMatrix, "ComplexMatrix");
timedReport("MatrixPowerComplex", 'proc(x) x^5; end proc', 1/2*maxMatrix, "ComplexMatrix");
timedReport("MeanComplex", '(Student[Statistics]:-Mean)', 10 * maxVector, "ComplexVector");
timedReport("DetComplex", 'Determinant', maxMatrix, "ComplexMatrix");
timedReport("EigenvaluesComplex", 'Eigenvalues', maxMatrix, "ComplexMatrix");
timedReport("EigenvectorsComplex", 'Eigenvectors', maxMatrix, "ComplexMatrix");
timedReport("TransposeComplex", 'Transpose', maxMatrix, "ComplexMatrix");
timedReport("FlattenComplex", 'proc(m) convert(m, Vector[row]); end proc', maxMatrix, "ComplexMatrix");
timedReport("ElementPowerComplex", 'proc(m) map(proc(x) x^5; end proc, m); end proc', 1/2 * maxMatrix, "ComplexMatrix");
timedReport("MatrixExpComplex", 'MatrixExponential', maxMatrix / 2, "ComplexMatrix");
timedReport("MatrixRankComplex", 'LinearAlgebra:-Rank', 3/2 * maxMatrix, "ComplexMatrix");
```

```
timedReport("LUDecompositionComplex", 'LUDecomposition', maxMatrix, "ComplexMatrix");
timedReport("LeastSquaresComplex", 'proc(x) LeastSquares(x, Column(x, 1)); end proc', maxMatrix, "ComplexMatrix");
timedReport("AddOneComplex", 'proc(m) m+~1 end proc', maxMatrix, "ComplexMatrix");
timedReport("FourierComplex8", 'FourierTransform', 5 * maxVector, "ComplexVector8");
timedReport("MeanComplex8", '(Student[Statistics]:-Mean)', 10 * maxVector, "ComplexVector8");
timedReport("DotComplex8", 'proc(x) x . x; end proc', maxMatrix, "ComplexMatrix8");
timedReport("InverseComplex8", 'MatrixInverse', maxMatrix, "ComplexMatrix8");
timedReport("LinearSolveComplex8", 'proc(x) LinearSolve(x, sort(Column(x, 1))); end proc', maxMatrix, "ComplexMatrix8");
timedReport("CholeskyComplex8", 'proc(S) LUDecomposition((map(conjugate, Transpose(S))) . S, method = 'Cholesky'); end proc', maxMatrix,
"ComplexMatrix8");
timedReport("MatrixPowerComplex8", 'proc(x) x^5; end proc', 1/2*maxMatrix, "ComplexMatrix8");
timedReport("DetComplex8", 'Determinant', maxMatrix, "ComplexMatrix8");
timedReport("AddOneComplex8", 'proc(m) m+~1 end proc', maxMatrix, "ComplexMatrix8");
timedReport("EigenvaluesComplex8", 'Eigenvalues', maxMatrix, "ComplexMatrix8");
timedReport("EigenvectorsComplex8", 'Eigenvectors', maxMatrix, "ComplexMatrix8");
timedReport("TransposeComplex8", 'Transpose', maxMatrix, "ComplexMatrix8");
timedReport("FlattenComplex8", 'proc(m) convert(m, Vector[row]); end proc', maxMatrix, "ComplexMatrix8");
timedReport("ElementPowerComplex8", 'proc(m) map(proc(x) x^5; end proc, m); end proc', 1/4*maxMatrix, "ComplexMatrix8");
timedReport("MatrixExpComplex8", 'MatrixExponential', maxMatrix/2, "ComplexMatrix8");
timedReport("MatrixRankComplex8", 'LinearAlgebra:-Rank', 3/2 * maxMatrix, "ComplexMatrix8");
timedReport("LUDecompositionComplex8", 'LUDecomposition', maxMatrix, "ComplexMatrix8");
timedReport("LeastSquaresComplex8", 'proc(x) LeastSquares(x, sort(Column(x, 1))); end proc', maxMatrix, "ComplexMatrix8");
timedReport("DotInteger", 'proc(x) x . x; end proc', (1/2) * maxMatrix, "IntegerMatrix");
timedReport("InverseInteger", 'MatrixInverse', 1/20*maxMatrix, "IntegerMatrix");
timedReport("LinearSolveInteger", 'proc(x) LinearSolve(x, sort(Column(x, 1))); end proc', 1/5 * maxMatrix, "IntegerMatrix");
timedReport("MatrixPowerInteger", 'proc(x) x^5; end proc', 1/5 * maxMatrix, "IntegerMatrix");
timedReport("DetInteger", 'Determinant', 700, "IntegerMatrix");
timedReport("TransposeInteger", 'Transpose', maxMatrix, "IntegerMatrix");
timedReport("FlattenInteger", 'proc(m) convert(m, Vector[row]); end proc', maxMatrix, "IntegerMatrix");
timedReport("MeanInteger", '(Student[Statistics]):-Mean', maxVector, "IntegerVector");
timedReport("SortInteger", 'sort', maxVector, "IntegerVector");
timedReport("EigenvaluesInteger", 'Eigenvalues', 1/30*maxMatrix, "IntegerMatrix");
timedReport("ElementPowerInteger", 'proc(m) map(proc(x) x^5; end proc, m); end proc', maxMatrix, "IntegerMatrix");
timedReport("AddOneInteger", 'proc(m) m+~1 end proc', maxMatrix, "IntegerMatrix");
timedReport("DotInteger8", 'proc(x) x . x; end proc', (1/2) * maxMatrix, "IntegerMatrix8");
timedReport("LinearSolveInteger8", 'proc(x) LinearSolve(x, sort(Column(x, 1))); end proc', 1/4*maxMatrix, "IntegerMatrix8");
timedReport("MatrixPowerInteger8", 'proc(x) x^5; end proc', 1/4*maxMatrix, "IntegerMatrix8");
timedReport("DetInteger8", 'Determinant', 700, "IntegerMatrix8");
timedReport("TransposeInteger8", 'Transpose', maxMatrix, "IntegerMatrix8");
timedReport("FlattenInteger8", 'proc(m) convert(m, Vector[row]); end proc', maxMatrix, "IntegerMatrix8");
timedReport("MeanInteger8", '(Student[Statistics]:-Mean)', 10 * maxVector, "IntegerVector8");
timedReport("SortInteger8", 'sort', maxVector, "IntegerVector8");
timedReport("EigenvaluesInteger8", 'Eigenvalues', 1/30*maxMatrix, "IntegerMatrix8");
timedReport("ElementPowerInteger8", 'proc(m) map(proc(x) x^5; end proc, m); end proc', maxMatrix, "IntegerMatrix8");
timedReport("AddOneInteger8", 'proc(m) m+~1 end proc', maxMatrix, "IntegerMatrix8");
timedReport("SortCustomFloat8", 'proc(data) sort(data, proc(x, y) abs(y) < abs(x); end proc); end proc', maxVector, "VectorFloat8");
timedReport("SortCustomReal", 'proc(data) sort(data, proc(x, y) abs(y) < abs(x); end proc); end proc', maxVector, "Vector");
timedReport("SortCustomComplex", 'proc(data) sort(data, proc(x, y) abs(y) < abs(x); end proc); end proc', maxVector, "ComplexVector");
timedReport("SortCustomComplex8", 'proc(data) sort(data, proc(x, y) abs(y) < abs(x); end proc); end proc', maxVector, "ComplexVector8");
timedReport("SortCustomInteger", 'proc(data) sort(data, proc(x, y) abs(y) < abs(x); end proc); end proc', maxVector, "IntegerVector");
timedReport("SortCustomInteger8", 'proc(data) sort(data, proc(x, y) abs(y) < abs(x); end proc); end proc', 100000, "IntegerVector8");
timedReport("MatrixRankInteger", 'LinearAlgebra:-Rank', maxMatrix/5, "IntegerMatrix");
timedReport("MatrixRankInteger8", 'LinearAlgebra:-Rank', maxMatrix/5, "IntegerMatrix8");

evaluateTest(
 "RandomNumbers",
 [
  ["RandomReal", 'makeData("Matrix", 2000)'],
  ["RandomInteger", 'makeData("IntegerMatrix", 2000)'],
  ["RandomExtended", 'makeData("BigNumberMatrix", 2000)'],
```

```
  ["RandomExtended1000", 'makeData("BigNumberMatrix1000", 2000)'],
  ["RandomNormal", 'Sample(RandomVariable(Normal(0, 1)), 10^7)'],
  ["RandomPoisson", 'Sample(RandomVariable(Poisson(4)), 10^7)'],
  ["RandomBinomial", 'Sample(RandomVariable(BinomialDistribution(10, 0.2)), 10^7)'],
  ["RandomHypergeometric", 'Sample(RandomVariable(HypergeometricDistribution(100, 50, 50)), 10^7)'],
  ["RandomGeometric", 'Sample(RandomVariable(GeometricDistribution(1/3)), 10^7)']
  ]
 );

timedReport("SingularValuesReal", 'SingularValues', 2 * maxMatrix, "Matrix");
timedReport("SingularValuesFloat8", 'SingularValues', maxMatrix, "MatrixFloat8");
timedReport("SingularValuesComplex", 'SingularValues', maxMatrix, "ComplexMatrix");
timedReport("SingularValuesComplex8", 'SingularValues', maxMatrix, "ComplexMatrix8");
timedReport("SingularValuesInteger", 'SingularValues', maxMatrix/50, "IntegerMatrix");
timedReport("SingularValuesInteger8", 'SingularValues', maxMatrix/50, "IntegerMatrix8");
timedReport("QRDecompositionReal", 'SingularValues', maxMatrix, "Matrix");
timedReport("QRDecompositionFloat8", 'SingularValues', maxMatrix, "MatrixFloat8");
timedReport("QRDecompositionComplex", 'QRDecomposition', maxMatrix, "ComplexMatrix");
timedReport("QRDecompositionComplex8", 'QRDecomposition', maxMatrix, "ComplexMatrix8");
timedReport("QRDecompositionInteger", 'QRDecomposition', maxMatrix/50, "IntegerMatrix");
timedReport("QRDecompositionInteger8", 'QRDecomposition', maxMatrix/50, "IntegerMatrix8");

numerictest("ElementaryFunctionsReal", fnList, maxVector, "Vector");
numerictestelementwise("ElementaryFunctionsElementwiseReal", elementwisefnList, maxVector, "Vector");
numerictestinplace("ElementaryFunctionsInPlaceReal", fnList, maxVector, "Vector");

numerictest("ElementaryFunctionsComplex", fnList, maxVector, "ComplexVector");
numerictestinplace("ElementaryFunctionsInPlaceComplex", fnList, maxVector, "ComplexVector");
numerictestelementwise("ElementaryFunctionsElementwiseComplex", elementwisefnList, maxVector, "ComplexVector");

numerictest("ElementaryFunctionsFloat8", fnList, 10 * maxVector, "VectorFloat8");
numerictestinplace("ElementaryFunctionsInPlaceFloat8", fnList, 10 * maxVector, "VectorFloat8");
numerictestelementwise("ElementaryFunctionsElementwiseFloat8", elementwisefnList, 10 * maxVector, "VectorFloat8");

numerictest("ElementaryFunctionsComplex8", fnList, maxVector, "ComplexVector8");
numerictestinplace("ElementaryFunctionsInPlaceComplex8", fnList, maxVector, "ComplexVector8");
numerictestelementwise("ElementaryFunctionsElementwiseComplex8", elementwisefnList, maxVector, "ComplexVector8");

numerictestinplace("ElementaryFunctionsInPlaceSparse", fnList, 100 * maxSparseVector, "SparseVector");
numerictest("ElementaryFunctionsSparse", fnListZeroSafe, maxSparseVector, "SparseVector");
numerictestelementwise("ElementaryFunctionsElementwiseSparse", elementwisefnListZeroSafe, maxSparseVector, "SparseVector");

timedReport("RowReduceReal", 'ReducedRowEchelonForm', maxMatrix/5, "Matrix");
timedReport("RowReduceComplex", 'ReducedRowEchelonForm', maxMatrix/5, "ComplexMatrix");
timedReport("RowReduceInteger", 'ReducedRowEchelonForm', maxMatrix/10, "IntegerMatrix");
timedReport("RowReduceFloat8", 'ReducedRowEchelonForm', maxMatrix/5, "MatrixFloat8");
timedReport("RowReduceComplex8", 'ReducedRowEchelonForm', maxMatrix/5, "ComplexMatrix8");
timedReport("RowReduceInteger8", 'ReducedRowEchelonForm', maxMatrix/10, "IntegerMatrix8");

timedReport("NullSpaceReal", 'NullSpace', maxMatrix, "Matrix");
timedReport("NullSpaceComplex", 'NullSpace', maxMatrix, "ComplexMatrix");
timedReport("NullSpaceInteger", 'NullSpace', maxMatrix/10, "IntegerMatrix");
timedReport("NullSpaceFloat8", 'NullSpace', maxMatrix, "MatrixFloat8");
timedReport("NullSpaceComplex8", 'NullSpace', maxMatrix, "ComplexMatrix8");
timedReport("NullSpaceInteger8", 'NullSpace', maxMatrix/10, "IntegerMatrix8");

timedReport("ScalarMultiplyReal", 'proc(m) ScalarMultiply(m, 3); end proc', maxMatrix, "Matrix");
timedReport("ScalarMultiplyComplex", 'proc(m) ScalarMultiply(m, 3); end proc', maxMatrix, "ComplexMatrix");
timedReport("ScalarMultiplyInteger", 'proc(m) ScalarMultiply(m, 3); end proc', maxMatrix, "IntegerMatrix");
timedReport("ScalarMultiplyFloat8", 'proc(m) ScalarMultiply(m, 3); end proc', maxMatrix, "MatrixFloat8");
```

```
timedReport("ScalarMultiplyComplex8", 'proc(m) ScalarMultiply(m, 3); end proc', maxMatrix, "ComplexMatrix8");
timedReport("ScalarMultiplyInteger8", 'proc(m) ScalarMultiply(m, 3); end proc', maxMatrix, "IntegerMatrix8");

timedReport("DeleteRowReal", 'proc(m) DeleteRow(m, 3); end proc', maxMatrix, "Matrix");
timedReport("DeleteRowComplex", 'proc(m) DeleteRow(m, 3); end proc', maxMatrix, "ComplexMatrix");
timedReport("DeleteRowInteger", 'proc(m) DeleteRow(m, 3); end proc', maxMatrix, "IntegerMatrix");
timedReport("DeleteRowFloat8", 'proc(m) DeleteRow(m, 3); end proc', maxMatrix, "MatrixFloat8");
timedReport("DeleteRowComplex8", 'proc(m) DeleteRow(m, 3); end proc', maxMatrix, "ComplexMatrix8");
timedReport("DeleteRowInteger8", 'proc(m) DeleteRow(m, 3); end proc', maxMatrix, "IntegerMatrix8");

timedReport("DeleteColumnReal", 'proc(m) DeleteColumn(m, 3); end proc', maxMatrix, "Matrix");
timedReport("DeleteColumnComplex", 'proc(m) DeleteColumn(m, 3); end proc', maxMatrix, "ComplexMatrix");
timedReport("DeleteColumnInteger", 'proc(m) DeleteColumn(m, 3); end proc', maxMatrix, "IntegerMatrix");
timedReport("DeleteColumnFloat8", 'proc(m) DeleteColumn(m, 3); end proc', maxMatrix, "MatrixFloat8");
timedReport("DeleteColumnComplex8", 'proc(m) DeleteColumn(m, 3); end proc', maxMatrix, "ComplexMatrix8");
timedReport("DeleteColumnInteger8", 'proc(m) DeleteColumn(m, 3); end proc', maxMatrix, "IntegerMatrix8");


evaluateTest("HighPrecision", [
 ["Pi", 'evalf(Pi, 10^6)'],
 ["Sqrt", 'evalf(sqrt(2), 10^6)'],
 ["Sin", 'evalf(sin(1) + 1/(1 + sin(1)), 10^6)'],
 ["Exp", 'evalf(exp(1), 10^6)'],
 ["Log", 'evalf(log(5), 10^6)'],
 ["Tan", 'evalf(tan(1 + I), 10^6)'],
 ["BesselK", 'evalf(BesselK(2, 3), 5000)'],
 ["Gamma", 'evalf(GAMMA(11/3), 5000)'],
 ["GoldenRatio", 'evalf((1 + sqrt(5))/2, 10^6)'],
 ["Catalan",'evalf(Catalan,50000)'],
 ["EulerGamma",'evalf(gamma,50000)'],
 ["PlasticConstant",'evalf(RootOf(_Z^3-_Z-1),10^6)'],
 ["Glaisher", 'evalf(exp(1/12 - D(Zeta)(-1)), 5000)']
]):

Digits := 50;
timedReport("DotExtended", 'proc(x) x . x; end proc', 1/25*maxMatrix, "BigNumberMatrix");
timedReport("InverseExtended", 'MatrixInverse', 1/25*maxMatrix, "BigNumberMatrix");
timedReport("EigenvaluesExtended", 'Eigenvalues', 60, "BigNumberMatrix");
timedReport("LinearSolveExtended", 'proc(x) LinearSolve(x, sort(Column(x, 1))); end proc', 1/20 * maxMatrix, "BigNumberMatrix");
timedReport("CholeskyExtended", 'proc(S) LUDecomposition((Transpose(S)) . S, method = 'Cholesky'); end proc', 1/20*maxMatrix, "BigNumberMatrix");
timedReport("MatrixPowerExtended", 'proc(x) x^5; end proc', 1/25*maxMatrix, "BigNumberMatrix");
timedReport("DetExtended", 'Determinant', 1/10*maxMatrix, "BigNumberMatrix");
timedReport("TransposeExtended", 'Transpose', maxMatrix, "BigNumberMatrix");
timedReport("FlattenExtended", 'proc(m) convert(m, Vector[row]); end proc', maxMatrix, "BigNumberMatrix");
timedReport("SortExtended", 'sort', maxVector, "BigNumberVector");
timedReport("MeanExtended", '(Student[Statistics]:-Mean)', 1/10*maxVector, "BigNumberVector");
timedReport("EigenvectorsExtended", 'Eigenvectors', 1/25*maxMatrix, "BigNumberMatrix");
timedReport("ElementPowerExtended", 'proc(m) map(proc(x) x^5; end proc, m); end proc', 1/2*maxMatrix, "BigNumberMatrix");
timedReport("MatrixExpExtended", 'MatrixExponential', 50, "BigNumberMatrix");
timedReport("SortCustomExtended", 'proc(data) sort(data, proc(x, y) abs(y) < abs(x); end proc); end proc', 2*maxVector, "BigNumberVector");
timedReport("MatrixRankExtended", 'LinearAlgebra:-Rank', maxMatrix/10, "BigNumberMatrix");
timedReport("AddOneExtended", 'proc(m) m+~1 end proc', maxMatrix/2, "BigNumberMatrix");
numerictest("ElementaryFunctionsExtended", fnList, 1/10*maxVector, "BigNumberVector");
numerictestelementwise("ElementaryFunctionsElementwiseExtended", elementwisefnList, 1/10*maxVector, "BigNumberVector");
numerictestinplace("ElementaryFunctionsInPlaceExtended", fnList, 1/10*maxVector, "BigNumberVector");
timedReport("SingularValuesExtended", 'SingularValues', maxMatrix/10, "BigNumberMatrix");
timedReport("QRDecompositionExtended", 'QRDecomposition', maxMatrix/10, "BigNumberMatrix");
timedReport("LUDecompositionExtended", 'LUDecomposition', maxMatrix/10, "BigNumberMatrix");
timedReport("LeastSquaresExtended", 'proc(x) LeastSquares(x, sort(Column(x, 1))); end proc', 1/25*maxMatrix, "BigNumberMatrix");
timedReport("RowReduceExtended", 'ReducedRowEchelonForm', maxMatrix/10, "BigNumberMatrix");
```

```
timedReport("NullSpaceExtended", 'NullSpace', maxMatrix/40, "BigNumberMatrix");
timedReport("ScalarMultiplyExtended", 'proc(m) ScalarMultiply(m, 3); end proc', maxMatrix, "BigNumberMatrix");
timedReport("DeleteRowExtended", 'proc(m) DeleteRow(m, 3); end proc', maxMatrix, "BigNumberMatrix");
timedReport("DeleteColumnExtended", 'proc(m) DeleteColumn(m, 3); end proc', maxMatrix, "BigNumberMatrix");

Digits := 1000;
timedReport("DotExtended1000", 'proc(x) x . x; end proc', 60, "BigNumberMatrix1000");
timedReport("InverseExtended1000", 'MatrixInverse', 1/25 * maxMatrix, "BigNumberMatrix1000");
timedReport("EigenvaluesExtended1000", 'Eigenvalues', 60, "BigNumberMatrix1000");
timedReport("LinearSolveExtended1000", 'proc(x) LinearSolve(x, sort(Column(x, 1))); end proc', 1/25*maxMatrix, "BigNumberMatrix1000");
timedReport("CholeskyExtended1000", 'proc(S) LUDecomposition((Transpose(S)) . S, method = 'Cholesky'); end proc', 1/20*maxMatrix,
"BigNumberMatrix1000");
timedReport("MatrixPowerExtended1000", 'proc(x) x^5; end proc', 1/25*maxMatrix, "BigNumberMatrix1000");
timedReport("DetExtended1000", 'Determinant', 1/10*maxMatrix, "BigNumberMatrix1000");
timedReport("TransposeExtended1000", 'Transpose', maxMatrix, "BigNumberMatrix1000");
timedReport("FlattenExtended1000", 'proc(m) convert(m, Vector[row]); end proc', maxMatrix, "BigNumberMatrix1000");
timedReport("SortExtended1000", 'sort', maxVector, "BigNumberVector1000");
timedReport("MeanExtended1000", '(Student[Statistics]:-Mean)', 1/10 * maxVector, "BigNumberVector1000");
timedReport("EigenvectorsExtended1000", 'Eigenvectors', 1/25*maxMatrix, "BigNumberMatrix1000");
timedReport("SingularValuesExtended1000", 'SingularValues', 150, "BigNumberMatrix1000");
timedReport("QRDecompositionExtended1000", 'QRDecomposition', 150, "BigNumberMatrix1000");
timedReport("ElementPowerExtended1000", 'proc(m) map(proc(x) x^5; end proc, m); end proc', 1/2*maxMatrix, "BigNumberMatrix1000");
timedReport("MatrixExpExtended1000", 'MatrixExponential', 24, "BigNumberMatrix1000");
timedReport("SortCustomExtended1000", 'proc(data) sort(data, proc(x, y) abs(y) < abs(x); end proc); end proc', maxVector, "BigNumberVector1000");
timedReport("MatrixRankExtended1000", 'LinearAlgebra:-Rank', maxMatrix/20, "BigNumberMatrix1000");
timedReport("AddOneExtended1000", 'proc(m) m+~1 end proc', maxMatrix/2, "BigNumberMatrix1000");
timedReport("LUDecompositionExtended1000", 'LUDecomposition', maxMatrix/10, "BigNumberMatrix1000");
timedReport("LeastSquaresExtended1000", 'proc(x) LeastSquares(x, sort(Column(x, 1))); end proc', 1/40 * maxMatrix, "BigNumberMatrix1000");
timedReport("RowReduceExtended1000", 'ReducedRowEchelonForm', maxMatrix/10, "BigNumberMatrix1000");
timedReport("NullSpaceExtended1000", 'NullSpace', maxMatrix/40, "BigNumberMatrix1000");
timedReport("ScalarMultiplyExtended1000", 'proc(m) ScalarMultiply(m, 3); end proc', maxMatrix, "BigNumberMatrix1000");
timedReport("DeleteRowExtended1000", 'proc(m) DeleteRow(m, 3); end proc', maxMatrix, "BigNumberMatrix1000");
timedReport("DeleteColumnExtended1000", 'proc(m) DeleteColumn(m, 3); end proc', maxMatrix, "BigNumberMatrix1000");
```