Maple 13 における効率性の改善点

Maple 13には、数多くの効率性改善が施されています。下記はその処理能力が特に向上したいくつかの機能に関する説明です。

Q および円分体の線形方程式系

- 新しい dense p-adic lift アルゴリズムにより、LinearAlgebra[LinearSolve] および LinearAlgebra[Modular][IntegerLinearSolve]の効率が改善されました。詳細については、 Enhancements to LinearAlgebra を参照してください。
- Maple 13 には、円分体の線形方程式系を解く高速モジュラーアルゴリズムを備えています。たとえば、線形方程式系において、$f(x)$が 次数 >1 の円周等分多項式である $RootOf(f,x)$ を含む場合、高速コードを使用します。 numtheory[cyclotomic]も参照してください。

比較として、次の例を検討します：

```
> r := RootOf(x^4+x^3+x^2+x+1);
```

$$r := RootOf(\_Z^4 + \_Z^3 + \_Z^2 + \_Z + 1) \quad (1.1)$$

```
> cf := ()->randpoly(r, degree = 3):
> nv := 50:
> vars := seq(x[i],i=1..nv):
> sys := {seq(randpoly([vars],degree=1,coeffs=cf),i=1..nv)}:
> tt := time():
> solve(sys,{vars}):
> time()-tt;
```

$$0.212 \quad (1.2)$$

Maple 12 では、同じ問題に約 7 倍の時間がかかっていました。

LinearAlgebra の効率をさらに改善

行列をコピーして輪郭を伴う形で結果にコピーする作業（インデックス機能）の処理速度が向上しました。次のコレスキー分解の演算は、Maple 12 の約 20 倍の速度で処理されます。

```
> with(LinearAlgebra):
> N := 1000:
  M := RandomMatrix(N,N,generator=-0.5..0.5, outputoptions=
  [datatype=float[8]]):
> L := Matrix(M,shape=triangular[lower]):
> M := Matrix(L.Transpose(L)):
> for i from 1 to N do M[i,i]:=i; end do:
> for i from 1 to N do M[i,i] := i; end do:
> st := time():
```

```
> LUDecomposition(M,method=Cholesky):
```

```
> time() - st;
```

0.02 (2.1)

次のUL分解の演算は、Maple 12 の約 10 倍の速度で処理されるだけでなく、メモリ割当ては Maple 12 の 80% です。

```
> with(LinearAlgebra):
```

```
> N := 1000:
  M := RandomMatrix(N,N,generator=-0.5..0.5, outputoptions=
  [datatype=float[8]]):
```

```
> st := time():
```

```
> LUDecomposition(M):
```

```
> time()-st;
```

0.03 (2.2)

SingularValues に新たに追加されたthin オプションにより、 $\min(m, n)$ の左および右特異ベクトルのみを伴う結果を出すことができます。そのため、行数が列数より大幅に多い場合に、LeastSquares の解で際立った高速化を実現します。次の特異値分解の演算は、Maple 12 で採用されていた thin オプション不使用の計算法と比べ、約 30 倍の速度で処理され、メモリ割当てはわずか 4% です。

```
> with(LinearAlgebra)
```

```
> m := 10000:
  n := 100:
  M := RandomMatrix(m,n,generator=-1.0..1.0, outputoptions=
  [datatype=float[8]]):
```

```
> st := time():
```

```
> SingularValues(M,output=[U,S,Vt],thin=true):
```

```
> time()-st;
```

1.78 (2.3)

モジュラー形式の多変量多項式の乗算および除算

新たにヒープベースのモジュラー形式の多変量多項式の乗除算アルゴリズムが追加され、これまでの乗除算の実行と比べ、処理速度が向上しました。
次の展開の処理速度は約 50% アップしました：

```
> p1 := randpoly([x,y,z],degree=20,terms=50):
```

```
> tt := time():
```

```
> p2 := modp(Expand(p1 ^ 3),13):
```

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

0.024 (3.1)

次の除算の処理速度は約 60% アップしました：

```
> tt := time():
```

```
> modp(Divide(p2,p1),13);
```

true (3.2)

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

0.019 (3.3)

上記コマンドに関する詳細については、 Expand および Divide を参照してください。

代数的数体および関数体の最大公約数を求める新しいスパースモジュラーアルゴリズム

特にスパースな多項式に効果的な、代数的数体および関数体の最大公約数を求めるアルゴリズムが新たに追加されました。たとえば、この Gcd の計算では処理速度が約 95% 向上しています：

```
> alias(p=RootOf(z^3+t*z^2+s*t,z)):
> alias(q=RootOf(z^2+2)):
> g := (y+p*t+q)*(t*x+s*y*p+q):
> a := t*x+s*y+p*s*t+p^2+q:
> b := x+s*x*y-t*p^2+p+t+q*s:
> A := evala(Expand(g*a)):
> B := evala(Expand(g*b)):
> tt := time():
> evala(Gcd(A, B));
> time()-tt;
```

0.5 (4.1)

lcoeff および tcoeff の効率改善

lcoeff および tcoeff の処理効率が向上しました。Maple 12では、次の最高次係数(leading coefficient)および最低次係数(trailing coefficient)の計算に、2倍以上の時間がかかり、使用メモリ量もかなり上回っていました：

```
> vars := [seq(x[i],i=0..19)]
```

$$vars := [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}] \quad (5.1)$$

```
> f := add(i*vars[i], i=1..20);
```

$$f := x_0 + 2x_1 + 3x_2 + 4x_3 + 5x_4 + 6x_5 + 7x_6 + 8x_7 + 9x_8 + 10x_9 + 11x_{10} + 12x_{11} + 13x_{12} + 14x_{13} + 15x_{14} + 16x_{15} + 17x_{16} + 18x_{17} + 19x_{18} + 20x_{19} \quad (5.2)$$

```
> g := expand(f^6):
> (nops,length)(g);
```

177100, 10915266 (5.3)

```
> t := time():
  (lcoeff,tcoeff)(g, vars);
  time() - t;
```

1, 64000000

0.316 (5.4)

これ以外の lcoeff および tcoeff の改良点については、 Enhancements to Symbolic Capabilities in Maple 13で説明しています。

heap[extract] 機能の効率改善

旧バージョンと比べ、 heap[extract] 機能で実行する比較数が少なくなっています。Maple 13 の場合と Maple 12 の場合について、要素 *n* のヒープからすべての要素を抽出する際に実行される比較数を以下の表に示します：

Number of Elements	Maple 12	Maple 13
10	27	26
20	89	68
30	158	116
40	239	170
50	338	242
60	416	295
70	517	345
80	642	416
90	737	483
100	841	554

マルチスレッド性能

評価エンジンの大部分がアップデートされ、パラレル処理を実行する際の性能が向上しました。さらに改良が必要な部分もありますが、パラレル処理性能は大幅に高速化しました。

```
> p := proc(A::Vector,B::Vector,C::Vector,m::integer,n::integer)
    option hfloat;
    local i;

    for i from m to n
    do
       C[i] := A[i]^2+B[i]^2;
    end do;
end proc:
n := 10^7:
m := n/2:
A := LinearAlgebra:-RandomVector( n, outputoptions=[datatype=
hfloat] ):
B := LinearAlgebra:-RandomVector( n, outputoptions=[datatype=
hfloat] ):
C := Vector( 1..n, datatype=hfloat ):
```

```
> tt := time[real]():
p(A,B,C,1,m):
p(A,B,C,m+1,n):
time[real]() - tt;
```

33.089 (7.1)

```
> tt := time[real]():
id1 := Threads:-Create( p(A,B,C,1,m) ):
```

```
p(A,B,C,m+1,n):
Threads:-Wait( id1 ):
time[real]() - tt;
```

22.873 (7.2)

Maple 12 では、パラレル処理の場合 35.568 秒かかり、シングルスレッドの場合より時間が若干長くかかっていました。

要素単位の演算(Element-wise operations)

elementwise 演算子構文は、場合によっては zip や map より効率的です。特定の演算については、zip や map で使用される一般的コードより最適化されています。それ以外の場合では、入力内容により、組込み済みハードウェアの浮動小数点ルーチンを直接使用することができます。

```
> A := LinearAlgebra:-RandomMatrix(1000):
```

```
> time(zip(`*`,A,2));
```

0.608 (8.1)

```
> time( A *~ 2 );
```

0.096 (8.2)

```
> B := LinearAlgebra:-RandomMatrix(1000,outputoptions=[datatype=
  float[8]]):
```

```
> time(zip(`*`,B,2));
```

```
> time( B *~ 2 );
```

0.017 (8.3)

```
> time(map(`sin`,B));
```

5.456 (8.4)

```
> time( sin~(B));
```

0.116 (8.5)

ImageTools パッケージ内の GetSubImage の改良点

ImageTools パッケージの GetSubImageルーチンは、出力が指定されている in-place のケースでは、処理速度が速くなります。次の GetSubImage 演算の繰り返しでは、処理速度が Maple 12 の約 2 倍となります。

```
> N := 500:
```

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

```
> image := Array(1..N,1..N,datatype=float[8],order=C_order):
```

```
> container := Array(1..n,1..n,datatype=float[8],order=C_order):
```

```
> st := time():
```

```
> for i from 1 to 1000 do
    ImageTools:-GetSubImage(image,10,10,n,n,output=container);
  end do:
```

```
> time() - st;
```

(9.1)

0.220 (9.1)

ガーベージコレクション頻度の削減

旧バージョンでは、10^6 ワードのメモリが割当てられると、ガーベージコレクションのサイクルが開始されていました。現在のコンピュータでは使用可能なメモリ量が増加していることから、ガーベージコレクションの頻度を下げても、メモリのサブシステムに多大な負荷をかけることはありません。Maple 13 では、最小で 10^7 ワードの割当て後にガーベージコレクション機能が起動します。この変更により、32-ビットのプロセッサでは平均 19%、64-ビットのプロセッサでは約 10% の改善が見られ、メモリ使用量の増加が緩やかになります。ガーベージコレクションの頻度は、kernelopts メカニズムを使用した gcfreq 変数の調節により、お使いのアプリケーションでベストパフォーマンスを引き出せるよう調整することが可能です。
以下に例を示します。

```
> with(LinearAlgebra):
```

```
> gcTest := proc()
          local A, n, resultList;

          for n to 30 do
            A := RandomMatrix(n,n);
            resultList:= LUDecomposition(A,output=['P','L',
  'U1','R']);
          end do;
  end proc:
```

```
> kernelopts(gcfreq=10^6);
```

[10000000, 0.1] (10.1)

```
> gc():
  start := time():
  gcTest():
  t1 := time() - start;
```

t1 := 0.924 (10.2)

```
> kernelopts(gcfreq=10^7);
```

1000000 (10.3)

```
> gc():
  start := time():
  gcTest():
  t2:= time() - start;
```

t2 := 0.860 (10.4)

参照

Index of New Maple 13 Features