# Polynomial factorization in Maple 2019

Michael Monagan and Baris Tuncer

Department of Mathematics, Simon Fraser University
mmonagan@cecm.sfu.ca

**Extended Abstract**

Maple 2019 has a new multivariate polynomial factorization algorithm for factoring polynomials in $\mathbb{Z}[x_1, x_2, ..., x_n]$, that is, polynomials in $n$ variables with integer coefficients. The new algorithm, which we call MTSHL, was developed by the authors at Simon Fraser University. The algorithm and its sub-algorithms have been published in a sequence of papers [3–5]. It was integrated into the Maple library in early 2018 by Baris Tuncer under a MITACS internship with Maplesoft. MTSHL is now the default factoring algorithm in Maple 2019.

The multivariate factorization algorithm in all previous versions of Maple is based mainly on the work of Paul Wang in [6, 7]. Keith Geddes is the main author of the Maple code. The algorithm and sub-algorithms are described in Chapter 6 of [1]. Wang's algorithm is still available in Maple 2019 with the `method="Wang"` option to the factor command.

Wang's method can be exponential in $n$ the number of variables. MTSHL is a random polynomial time algorithm. In [3] we found that it is faster than previous polynomial time methods of Kaltofen [2] and Zippel [8] and competitive with Wang's method in cases where Wang's method is not exponential in $n$.

Here we give an overview of the main idea in MTSHL. Let $a$ be the input polynomial to be factored. Suppose $a = fg$ for two irreducible factors $f, g \in \mathbb{Z}[x_1, \ldots, x_n]$. The multivariate polynomial factorization algorithm used in all computer algebra systems is based on Multivariate Hensel Lifting (MHL). For a description of MHL see Chapter 6 of [1]. MHL first chooses integers $\alpha_2, \alpha_3, \ldots, \alpha_n$ that satisfy certain conditions and factors the univariate image $a_1 = a(x_1, \alpha_2, \ldots, \alpha_n)$ in $\mathbb{Z}[x_1]$. Suppose $a_1(x_1) = f_1(x_1)g_1(x_1)$ and $f_1(x_1) = f(x_1, \alpha_2, \ldots, \alpha_n)$ and $g_1(x_1) = g(x_1, \alpha_2, \ldots, \alpha_n)$. Next MHL begins Hensel lifting. Wang's design of Hensel lifting recovers the variables $x_2, \ldots, x_n$ in the factors $f$ and $g$ one at a time in a loop. Let us use the notation

$$f_j = f(x_1, \ldots, x_j, \alpha_{j+1}, \ldots, \alpha_n) \quad \text{for} \quad j \geq 1.$$

So at the $j$'th step of MHL we have the factorization $a_{j-1} = f_{j-1}g_{j-1}$ and we want to obtain the factorization $a_j = f_j g_j$. Consider the polynomials $f_j$ and $g_j$ expanded as a Taylor polynomial about $x_j = \alpha_j$

$$f_j = \sum_{i=0}^{\deg(f_j, x_j)} \sigma_i (x_j - \alpha_j)^i \quad \text{and} \quad g_j = \sum_{i=0}^{\deg(g_j, x_j)} \tau_i (x_j - \alpha_j)^i$$

Here $\sigma_i, \tau_i \in \mathbb{Z}[x_1, \ldots, x_{j-1}]$ and $\sigma_0 = f_{j-1}$ and $\tau_0 = g_{j-1}$ are known. MHL recovers the coefficients $\sigma_i, \tau_i$ one at a time in a loop. Let $\text{supp}(\sigma)$ denote the support of $\sigma$, that is, the set of monomials in $\sigma$. Before continuing we give an example to fix the ideas and notation presented so far. Let $f = x_1^3 - x_1x_2x_3^2 + x_2^3x_3^2 + x_3^3 - 27$. For $\alpha_3 = 2$, we have $f_2 = f(x_1, x_2, 2) = x_1^3 + 4\,x_2^3 - 4\,x_1x_2 - 19$. Expanding $f$ about $x_3 = 2$ we have

$$f = \underbrace{(x_1^3 + 4\,x_2^3 - 4\,x_1x_2 - 19)}_{\sigma_0} + \underbrace{(4\,x_2^3 - 4\,x_1x_2 + 12)}_{\sigma_1}(x_3 - 2) + \underbrace{(x_2^3 - x_1x_2 + 6)}_{\sigma_2}(x_3 - 2)^2 + \underbrace{1}_{\sigma_3}(x_3 - 2)^4.$$

We have $\text{supp}(\sigma_0) = \{x_1^3, x_2^3, x_1x_2, 1\}$, $\text{supp}(\sigma_1) = \text{supp}(\sigma_2) = \{x_2^3, x_1x_2, 1\}$, and $\text{supp}(\sigma_3) = \{1\}$. Multivariate Hensel Lifting (MHL) computes $\sigma_i$ and $\tau_i$ by solving the multivariate polynomial diophantine (MDP) equation

$$\sigma_i g_{j-1} + \tau_i f_{j-1} = c_i \ \text{ in } \ \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$$

where the polynomial $c_i$ is the Taylor coefficient

$$\text{coeff}\Big(\, a_j - \left(\sum_{i=0}^{k-1} \sigma_i (x_j - \alpha_j)^i\right)\left(\sum_{i=0}^{k-1} \tau_i (x_j - \alpha_j)^i\right),\ (x_j - \alpha_j)^k \,\Big).$$

Most of the time in MHL is solving these MDP equations. Wang's method for solving them is recursive. If the $\alpha_2, \ldots, \alpha_n$ are non-zero Wang's method is exponential in $n$. For many polynomials it is possible to use zero for some or all $\alpha_j$ and avoid this exponential behaviour. But this is not always possible as there are several conditions that $\alpha_2, \ldots, \alpha_n$ must satisfy. The sparse Hensel lifting methods of [2] and [8] were developed to solve this problem. It turns out that if the integer $\alpha_j$ is chosen randomly from a large set then

$$\text{supp}(\sigma_i) \supseteq \text{supp}(\sigma_{i+1}) \ \text{ for } \ 0 \le i < \deg(f_j, x_j) \tag{1}$$

with high probability. The reader may verify this support chain holds in Example 1 where $\alpha = 2$ but it does not hold if $\alpha = 0$. See Lemma 1 in [3] for a precise statement for the probability and proof. MTSHL exploits (1) by using $\text{supp}(\sigma_{i-1})$ as the support for $\sigma_i$ to construct linear systems to solve for the coefficients of $\sigma_i$ in $x_1$. The linear systems are $t_j \times t_j$ transposed Vandermonde systems where $t_j = \#\text{coeff}(\sigma_i, x_1^j)$. We use Zippel's method from [9] to solve them in $O(t_j^2)$ time and $O(t_j)$ space. Since the number of terms in $\sigma_i$ and $\tau_i$ is not more than those in $f$ and $g$ respectively, our algorithm takes advantage of sparse factors $f$ and $g$.

We present two benchmarks comparing the new algorithm MTSHL in Maple 2019 with Wang's method in Maple 2019. The following Maple code creates an input polynomial $a \in \mathbb{Z}[x_1, \ldots, x_n]$ which is a product of two factors $f \times g$. Each factor has $n$ variables, 100 terms, and degree at most $d$. The Maple command `randpoly` creates each term randomly to have degree at most $d$ with an integer coefficient chosen at random from $[-10^6, 10^6]$.

```
kernelopts(numcpus=1); t := 100; d := 15;
for n from 5 to 12 do
  X := [seq( x||i, i=1..n )];
  f := randpoly(X,coeffs=rand(-10^6..10^6),terms=100,degree=d);
  g := randpoly(X,coeffs=rand(-10^6..10^6),terms=100,degree=d);
  a := expand(f*g);
  h := CodeTools[Usage]( factor(a,method="Wang") );
  #h := CodeTools[Usage]( factor(a) ); # Uses MTSHL in Maple 2019
od:
```

| $n$ | Wang (MDP) | MTSHL | $n$ | Wang (MDP) | MTSHL |
|---|---|---|---|---|---|
| 5 | 4.87s (89.4%) | .509s | 10 | 65.55s (98.0%) | .911s |
| 6 | 8.67s (85.8%) | .589s | 11 | 154.8s (98.0%) | .989s |
| 7 | 6.77s (91.2%) | .616s | 12 | 169.8s (99.0%) | 1.78s |
| 8 | 35.04s (94.7%) | .718s | 13 | 163.8s (96.5%) | 1.16s |
| 9 | 40.33s (99.6%) | .788s | 14 | 603.6s (98.7%) | 2.37s |

**Table 1.** Factorization timings in CPU seconds

Shown in column (MDP) in Table 1 is the percentage of time Wang's algorithm spent solving Multivariate Diophantine Equations. MTSHL is not impacted significantly by the number of variables. In theory the cost of MTSHL is linear in $n$ which is supported by this example.

Let $C_n$ denote the $n \times n$ cyclic matrix. See Figure 1. Observe that $\det C_n$ is a homogeneous polynomial in $\mathbb{Z}[x_1, \ldots, x_n]$. Because the factors of $\det C_n$ are dense, MTSHL has no inherent advantage over Wang's method and we expected it to be slower than Wang's method.

$$\begin{bmatrix} x_1 & x_2 & \ldots & x_{n-1} & x_n \\ x_n & x_1 & \ldots & x_{n-2} & x_{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_3 & x_4 & \ldots & x_1 & x_2 \\ x_2 & x_3 & \ldots & x_n & x_1 \end{bmatrix}$$

$$\begin{aligned} &(x_1 + x_2 + x_3 + x_4) \\ &(x_1 - x_2 + x_3 - x_4) \\ &\left(x_1^2 - 2\,x_1\,x_3 + x_2^2 - 2\,x_2\,x_4 + x_3^2 + x_4^2\right) \end{aligned}$$

**Fig. 1.** The cyclic $n \times n$ matrix $C_n$ and the factors of $\det(C_4)$.

Maple code for computing $\det C_n$ and factoring $\det C_n$ is given below. Note, for a homogenous input polynomial $\det C_n$, the `factor` command evaluates one variable $x_i = 1$, factors $\det(C_n)(x_i = 1)$ then homogenizes the factors. To fix $i$ we compute and factor $\det(C_n(x_n = 1))$.

```
kernelopts(numcpus=1);
for n from 6 to 10 do
  Cn := Matrix(n,n,shape=Circulant[x]);
```

```
  Cn := eval(Cn,x[n]=1); # dehomogenize Cn
  d := LinearAlgebra[Determinant](Cn,method=minor);
  F := CodeTools[Usage](factor(d));
  #F := CodeTools[Usage](factor(d,method="Wang"));
od:
```

Table 2 contains data for $\det C_n$ and timing data for factoring $\det C_n(x_n = 1)$. Column 2 is the number of terms of $\det C_n$. Column 3 is the degrees of the factors of $C_n$. Column 4 is the number of terms of the largest factor. Columns 5–7 are the CPU time to factor $\det C_n$ using our new algorithm MTSHL in Maple 2019, Wang's algorithm in Maple 2019 and Wang's algorithm in the Magma computer algebra system.

| $n$ | #det | $\deg(f_i)$ | max $\#f_i$ | MTSHL | Wang (MDP) | Magma |
|---|---|---|---|---|---|---|
| 8 | 810 | 1,1,2,4 | 86 | 0.140s | 0.096s (52%) | 0.12s |
| 9 | 2704 | 1,2,6 | 1005 | 0.465s | 0.253s (76%) | 1.02s |
| 10 | 7492 | 1,1,4,4 | 715 | 3.03s | 1.020s (49%) | 10.97s |
| 11 | 32066 | 1,10 | 184756 | 1.33s | 12.43s (88%) | 142.85s |
| 12 | 86500 | 1,1,2,2,2,4 | 621 | 4.97s | 20.51s (65%) | 7575.14s |
| 13 | 400024 | 1,12 | 2704156 | 10.24s | 212.40s (88%) | 30,871.9s |
| 14 | 1366500 | 1,1,6,6 | 27132 | 666.0s | 1364.4s (68%) | $> 10^6$s |

**Table 2.** Timings (CPU time seconds) for factoring $\det(C_n(x_n = 1))$

## References


1. Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for Computer Algebra*, Kluwer Acad. Publ. (1992).
2. Kaltofen, E., Sparse Hensel lifting. Proc. EUROCAL '85, LNCS **204**: 4–17, Springer, (1985).
3. Michael Monagan and Baris Tuncer. Using Sparse Interpolation in Hensel Lifting. *Proceedings of CASC 2016*, LNCS **9890**, 381–400, Springer, (2016).
4. Michael Monagan and Baris Tuncer. Factoring multivariate polynomials with many factors and huge coefficients. *Proceedings of CASC 2018*, LNCS **11077**, 319–334, Springer, (2018).
5. Michael Monagan and Baris Tuncer. The complexity of sparse Hensel lifting and sparse polynomial factorization. To appear in *J. Symbolic Computation*, 2019.
6. Wang, P.S., Rothschild, L.P. Factoring multivariate polynomials over the integers. *Mathematics of Computation*, **29**(131): 935–950, (1975).
7. Wang, P.S. An improved Multivariate Polynomial Factoring Algorithm. *Mathematics of Computation*, **32**:1215–1231, (1978).
8. Zippel, R.E. Newton's iteration and the sparse Hensel algorithm. *Proc.* SYMSAC 1981, pp. 68–72, ACM (1981).
9. Zippel, R.E. Interpolating polynomials from their values. *J. Symbolic Comput.*, **9**(3):375–403, (1990).