Symbolic-Numeric Algebra for Polynomials

Kosaku Nagasaka

Symbolic-Numeric Algebra for Polynomials (SNAP) **is a prototype package that includes basic functions for computing approximate algebraic properties, such as the approximate GCD of polynomials. Our aim is to show how the unified tolerance mechanism we introduce in the package works. Using this mechanism, we can carry out approximate calculations under certified tolerances. In this article, we demonstrate the functionality of the currently released package (Version 0.2), which is downloadable from wwwmain.h.kobe-u.ac.jp/~nagasaka/research/snap/index.phtml.en.**

■ Introduction

Recently, there have been many results in the area of symbolic-numeric algorithms, especially for polynomials (for example, approximate GCD for univariate polynomials [1, 2, 3, 4] and approximate factorization for bivariate polynomials [5, 6, 7, 8]). We think those results have practical value, which should be combined and implemented into one integrated computer algebra system such as *Mathematica*. In fact, *Maple* has such a special package called *SNAP (Symbolic-Numeric Algorithms for Polynomials)*. Currently, ours is the only such package available for *Mathematica*.

We have been developing our *SNAP* package for *Mathematica*, which is an abbreviation for Symbolic-Numeric **Algebra** for Polynomials. We use algebra to mean continuous capabilities of approximate operations; for example, computing an approximate GCD between an empirical polynomial and the nearest singular polynomial computed by SNAP functions of another empirical polynomial. This continuous applicability is more important for practical computations than the number of algorithms that are already implemented, especially for average users.

Our aim is to provide practical implementations of SNAP functions with a unified tolerance mechanism for polynomials like *Mathematica*'s floating-point numbers or Kako and Sasaki's effective floating-point numbers [9]. Our idea is very simple. We only have to add new data structures representing such polynomials with tolerances and basic calculation routines and SNAP functions for the structures. At this time, only simple tolerance representations (l^2-norm, l^1-norm, and l^∞-norm) for coefficient vectors of polynomials and a few SNAP functions are implemented, but the package is an ongoing project (Version 1.0 should be released in March 2007).

Features that are new in Version 0.2 [10, 11], include:

- SNAP structures for multivariate polynomials that have more than one variable.
- Basic functions that can operate with multivariate polynomials.
- New compatibilities with the built-in functions `PolynomialReduce` and `D`.
- New SNAP functions, such as `CoprimeQ`, `AbsolutelyIrreducibleQ`, `SeparationBound`, `Factor`, and `FactorList`.

□ Difference Between Numerical and Algebraic Computations

Let us consider a typical numerical computation—finding numerical roots:

```
In[1]:= System`NSolve[x^2 - 1 == 0, x, 20]

Out[1]= {{x → -1.0000000000000000000}, {x → 1.0000000000000000000}}
```

We might assume that polynomials that are close have roots that are close.

```
In[2]:= System`NSolve[x^2 - 1.0000000000001`20 == 0, x, 20]

Out[2]= {{x → -1.0000000000000500000}, {x → 1.0000000000000500000}}
```

If this argument is not correct, the problem is called "numerically unstable." Hence, "numerical stability" of the given problem and "significant digits" of the calculated solutions are important in numerical computations. *Mathematica*'s accuracy and precision tracking system are very useful for this purpose and are based on the concept that the higher precision used, the closer the solutions.

Unfortunately, this concept may not be correct for algebraic computations. Let us consider a factorization:

```
In[3]:= System`Factor[(x + y) (x - y) + 0.0000000001``16]

Out[3]= 1.00000 (1.00000 × 10^-10 + 1.00000 x^2 - 1.00000 y^2)
```

We cannot factor this polynomial if we increase the precision of the computation because algebraic properties are generally not continuous. Forward and backward error analyses may not be the solution, though they can be of supplemental use. The previous polynomial, for example, can be either reducible or irreducible, and we may not be able to know which property is correct.

We note that there are two approaches to operating with empirical polynomials in an extreme instance—using inexact or exact approximations. For example, let us consider factoring bivariate polynomials. The following numerical polynomial is reducible if we rationalize its coefficients up to machine precision:

$$f(x, y) = -7. + 7.x^2 - 1.x^3 + 1.x^5 + 15.y + 7.xy - 1.x^2 y + 2.x^3 y + 1.x^4 y - 2.y^2 - 2.xy^2 + 1.x^3 y^2 + 2.xy^3 + 1.x^2 y^3 \quad (1)$$

```
In[4]:= System`Factor[-7 + 7 x^2 - x^3 + x^5 + 15 y + 7 x y -
          x^2 y + 2 x^3 y + x^4 y - 2 y^2 - 2 x y^2 + x^3 y^2 + 2 x y^3 + x^2 y^3]

Out[4]= (-1 + x^2 + 2 y + x y) (7 + x^3 - y + x y^2)
```

This factorization is fine. The problem is operating with the following polynomial (we added small perturbation terms to the previous polynomial):

$$f(x, y) = \\ -7. + 7.x^2 - 1.x^3 + 1.x^5 + 15.y + 7.x\,y - 1.x^2\,y + 2.x^3\,y + 1.x^4\,y - \\ 2.y^2 - 2.x\,y^2 + 1.x^3\,y^2 + 2.x\,y^3 + 1.x^2\,y^3 + 0.0000001\,x - 0.0000001 \tag{2}$$

In general, we may not know whether these perturbation terms are numerical errors or actually exist in the polynomial. The inexact approximation approach tries to factor numerically, regardless of significant digits or magnitude of errors. Although we can check its backward error after calculations, we cannot know if the backward error is globally minimized or if the number of factors is globally maximized. Moreover, there is a possibility that an algorithm cannot find appropriate factors even if there are approximate factors.

Therefore, to do algebraic operations, we have to guarantee the properties for all possible polynomials that are sufficiently close to the given polynomial. For example, assuming the precision is 16, the polynomial set of the polynomial (2) includes the following polynomials:

$$f(x, y) = -7.00000000000000001 + 7.x^2 - 1.x^3 + 1.x^5 + \\ 15.y + 7.x\,y - 1.x^2\,y + 2.x^3\,y + 1.x^4\,y - 2.y^2 - 2.x\,y^2 + \\ 1.x^3\,y^2 + 2.x\,y^3 + 1.x^2\,y^3 + 0.0000001\,x - 0.0000001 \tag{3}$$

$$f(x, y) = -6.99999999999999999 + 7.x^2 - 1.x^3 + 1.x^5 + \\ 15.y + 7.x\,y - 1.x^2\,y + 2.x^3\,y + 1.x^4\,y - 2.y^2 - 2.x\,y^2 + \\ 1.x^3\,y^2 + 2.x\,y^3 + 1.x^2\,y^3 + 0.0000001\,x - 0.0000001 \tag{4}$$

Using exact approximations tries to guarantee such properties. However, this approach is more difficult than the first, because we have to guarantee an algebraic property of an infinite number of polynomials that are sufficiently close to the given polynomial.

■ Tolerance Mechanisms and Implementations

First, we introduce a framework for our package, which includes SNAP structures and their implementations using `UpSetDelayed`, `Format`, and so forth.

Function	Purpose
`SNAP`	converts the given polynomial to its SNAP representation.
`Normal`	converts the given SNAP representation to its normal polynomial expression.
`SNAPQ`	tests whether the given expression is a SNAP structure or not.
`Tolerance`	returns the tolerance of the given SNAP structure.
`SetTolerance`	overwrites the tolerance of the given SNAP structure.
`TransformSNAP`	converts the given SNAP structure to the structure with the given scheme.
`Element`	tests whether the given polynomial is in the given SNAP structure or not.
`Expand`	has the same effect as the built-in function.
`ReplaceAll`	has the same effect as the built-in function.

Table 1. Basic functions.

□ Tolerance Mechanisms

We introduce the following data structures for empirical polynomials. They are very simple, but there is no system in which we can automatically use these structures in a way similar to *Mathematica*'s floating-point numbers or Kako and Sasaki's effective floating-point numbers [9]. These structures are different from those used in the previous version of this package [10, 11]. The latest version can operate on polynomials that have more than two variables (bivariate or multivariate polynomials).

Definition 1 (SNAP Structures). *We define the following SNAP structures (like sets of neighbors) for approximate polynomials for the given polynomial* $f(u_1, \ldots, u_r) \in \mathbb{C}[u_1, \ldots, u_r]$ *and tolerance* $\epsilon \in \mathbb{R}$*:*

$$\begin{aligned} P_p(f, \epsilon) = \{\overline{f} \mid \overline{f} \in \mathbb{C}[u_1, \ldots, u_r], \\ \deg_u \overline{f} = \deg_u f\ (\forall u \in \{u_1, \ldots, u_r\}),\ \|f - \overline{f}\|_p \le \epsilon\}, \end{aligned} \tag{5}$$

where $\|f\|_p$ *denotes coefficient vector norms for polynomials:*

$$\|f\|_p = (\Sigma_i |f_i|^p)^{\frac{1}{p}},\ f = \Sigma_i f_i u_1^{e_{i1}} \cdots u_r^{e_{ir}}. \tag{6}$$

In the rest of this article, $P_*(f, \epsilon)$ means any $P_2(f, \epsilon)$, $P_1(f, \epsilon)$, or $P_\infty(f, \epsilon)$.

Remark 1. You might think that the following definition is better than the previous definition:

$$\begin{aligned} P_*(f, \epsilon) = \{\overline{f} \mid \overline{f} \in \mathbb{C}[u_1, \ldots, u_r], \\ \deg_u \overline{f} \le \deg_u f\ (\forall u \in \{u_1, \ldots, u_r\}),\ \|f - \overline{f}\|_* \le \epsilon\}. \end{aligned} \tag{7}$$

However, this definition produces strange results. For example, let ϵ be an arbitrary small positive real number and f be the following polynomial:

$$f = f_n x^n + f_{n-1} x^{n-1} + \cdots + f_1 x + f_0,\ f_i \in \mathbb{C},\ |f_n| \le \epsilon. \tag{8}$$

For this polynomial, $P_*(f, \epsilon)$ includes polynomials that have n roots (counting multiplicities) and also polynomials that have at most $n - 1$ roots. Moreover, one might want to preserve more than total degree in the multivariate case (sometimes which is called triangle degree while it is called rectangle degree in the definition 1). Those degree models are too advanced for the current results in this research area and they cause unusual results for the known algorithms; hence, we use the expressions of Definition 1.

The coefficient vector norms for polynomials have properties similar to normal vector norms (see [12] for a discussion of these basic properties); hence, we have the following corollary.

Corollary 1. *We have the following properties:*

$$P_2(f,\ \epsilon) \subseteq P_1\left(f,\ \epsilon\sqrt{\Pi_{i=1}^{r}(e_i+1)}\right), \tag{9}$$

$$P_2(f,\ \epsilon) \subseteq P_\infty(f,\ \epsilon), \tag{10}$$

$$P_1(f,\ \epsilon) \subseteq P_2(f,\ \epsilon), \tag{11}$$

$$P_1(f,\ \epsilon) \subseteq P_\infty(f,\ \epsilon), \tag{12}$$

$$P_\infty(f,\ \epsilon) \subseteq P_2\left(f,\ \epsilon\sqrt{\Pi_{i=1}^{r}(e_i+1)}\right), \tag{13}$$

$$P_\infty(f,\ \epsilon) \subseteq P_1\left(f,\ \epsilon\,\Pi_{i=1}^{r}(e_i+1)\right), \tag{14}$$

where e_i denotes the degree of $f(u_1, \ldots, u_r)$ with respect to u_i.

Lemma 1. *We have the following properties for polynomials $f(u_1, \ldots, u_r)$ and $h(u_1, \ldots, u_r)$:*

$$\begin{gathered}\forall\ \overline{f} \in P_2(f, \epsilon_f),\ \forall\ \overline{h} \in P_2(h, \epsilon_h),\\ \overline{f}\times\overline{h} \in\\ P_2\left(f\times h,\ \sqrt{\min\{\Pi_{i=1}^{r}(e_i+1), \Pi_{i=1}^{r}(d_i+1)\}}\ (\|f\|_2\,\epsilon_h + \|h\|_2\,\epsilon_f + \epsilon_f\,\epsilon_h)\right),\end{gathered} \tag{15}$$

$$\begin{gathered}\forall\ \overline{f} \in P_1(f, \epsilon_f),\ \forall\ \overline{h} \in P_1(h, \epsilon_h),\\ \overline{f}\times\overline{h} \in P_1(f\times h,\ \|f\|_1\,\epsilon_h + \|h\|_1\,\epsilon_f + \epsilon_f\,\epsilon_h),\end{gathered} \tag{16}$$

$$\begin{gathered}\forall\ \overline{f} \in P_\infty(f, \epsilon_f),\ \forall\ \overline{h} \in P_\infty(h, \epsilon_h),\\ \overline{f}\times\overline{h} \in\\ P_\infty(f\times h,\ \min\{\Pi_{i=1}^{r}(e_i+1), \Pi_{i=1}^{r}(d_i+1)\}\,(\|f\|_1\,\epsilon_h + \|h\|_1\,\epsilon_f + \epsilon_f\,\epsilon_h)),\end{gathered} \tag{17}$$

where e_i and d_i denote the degree of $f(u_1, \ldots, u_r)$ and $h(u_1, \ldots, u_r)$ with respect to u_i, respectively.

Proof of Lemma 1. The lemma is proved by the following properties of vector and matrix norms [12], since any multiplication of polynomials can be done as matrix multiplications:

$$\|A\|_2 \leq \|A\|_F, \tag{18}$$

$$\|A\|_1 = \max_{1 \leq j \leq n} \Sigma_{i=1}^{m} |a_{ij}|, \tag{19}$$

$$\|A\|_\infty = \max_{1 \leq i \leq m} \Sigma_{j=1}^{n} |a_{ij}|, \tag{20}$$

where $\|A\|_p$ denotes matrix p-norms for matrix A of size $m \times n$. □

Lemma 2. *If the following relation holds for polynomials* $f(u_1, \ldots, u_r)$ *and* $h(u_1, \ldots, u_r)$,

$$\begin{gathered} \forall\, \overline{f} \in P_p(f, \epsilon_f),\ \forall\, \overline{h} \in P_p(h, \epsilon_h), \\ \deg_u(\overline{f} + \overline{h}) = \max\{\deg_u f, \deg_u h\}\ (\forall\, u \in \{u_1, \ldots, u_r\}), \end{gathered} \tag{21}$$

we have the following properties:

$$\begin{gathered} \forall\, \overline{f} \in P_p(f, \epsilon_f),\ \forall\, \overline{h} \in P_p(h, \epsilon_h), \\ \overline{f} + \overline{h} \in P_p(f + h, \epsilon_f + \epsilon_h) \equiv \left\{ g \mid g \in \mathbb{C}[u_1, \ldots, u_r],\ \deg_u g = \right. \\ \left. \max\{\deg_u \overline{f}, \deg_u \overline{h}\}\ (\forall\, u \in \{u_1, \ldots, u_r\}),\ \|(f + h) - g\|_p \leq \epsilon_f + \epsilon_h \right\}, \end{gathered} \tag{22}$$

where e_i *and* d_i *denote the degree of* $f(u_1, \ldots, u_r)$ *and* $h(u_1, \ldots, u_r)$ *with respect to* u_i, *respectively, and* $p = 1, 2, \infty$.

Proof of Lemma 2. The lemma is directly proved by the triangle inequality for norms. □

The condition of Lemma 2 is necessary for ensuring equalities for degrees in $P_*(f + h, \epsilon_f + \epsilon_h)$. Without the condition, unusual results as in Remark 1 may occur. However, for example, we have to operate with the following computation and determine the set S. This is important especially for divisions (quotient and remainder) and reductions (for non-univariate polynomials):

$$\forall\, g \in P_*(f, \epsilon), \forall\, h \in P_*(f, \epsilon),\ g - h \in S. \tag{23}$$

We introduce a simple rule to handle this. If the norm of the leading coefficients of the representative polynomial f, with respect to u_i, is not larger than the tolerance ϵ, then we rewrite the SNAP structure as follows:

$$\begin{gathered} P_*(f, \epsilon) \Rightarrow P_*(f_{\text{new}}, \epsilon) = \{g \mid g \in \mathbb{C}[u_1, \ldots, u_r], \\ \deg_u g = \deg_u f_{\text{new}}\ (\forall\, u \in \{u_1, \ldots, u_r\}),\ \|f_{\text{new}} - g\|_2 \leq \epsilon\}, \end{gathered} \tag{24}$$

where f_{new} denotes f − (all the leading monomials of f, with respect to u_i).

□ Tolerance Implementations

According to the previous definitions, we have implemented the structures, basic operations (addition and multiplication) on the structures, and functions that transform one structure into another.

We define the following expression to represent the SNAP structures $P_2(f, \epsilon)$, $P_1(f, \epsilon)$, and $P_\infty(f, \epsilon)$, where `scheme` can be `AbsolutePolynomial2Norm`, `AbsolutePolynomial1Norm`, and `AbsolutePolynomialiNorm`, respectively. In a future release, other schemes will be implemented.

```
SNAP[f,ϵ,{{u1,e1}, ,{ur,er}},scheme]
```

Using this structure, we have implemented basic functions provided by the mechanisms and now show some of their expressions. We note that the package also provides functions that automatically transform the given polynomial to the previous representation.

```
TransformSNAP[SNAP[f_, epsf_, {{x_Symbol, n_}, yz___},
   AbsolutePolynomial2Norm], AbsolutePolynomialiNorm] :=
 SNAP[f, epsf, {{x, n}, yz}, AbsolutePolynomialiNorm]

Normal[SNAP[f_, epsf_, {{x_Symbol, n_}, yz___}, scheme_]] := f

SNAP /: -SNAP[f_, epsf_, {{x_Symbol, n_}, yz___}, scheme_] :=
 SNAP[-f, epsf, {{x, n}, yz}, scheme]

SNAP[g_, epsg_, varsg_List, scheme_] +
  SNAP[h_, epsh_, varsh_List, scheme_] ^:=
 SNAP[g + h, epsg + epsh, maxvarslist[varsg, varsh], scheme]

SNAP[g_, epsg_, varsg_, scheme_] + f_?NotSNAPQ ^:=
 SNAP[g, epsg, varsg, scheme] + SNAP[f]

Tolerance[SNAP[f_, epsf_, vars_, scheme_]] := epsf
```

One aim of the package is providing an environment in which we use SNAP structures transparently, like *Mathematica*'s floating-point numbers; hence, we have implemented the following `Format` code.

```
Format[SNAP[f_, e_, variables_, scheme_]] := N@f
```

In this version, we suppose that the basic four operations between *Mathematica*'s bigfloat numbers and its interval arithmetic guarantee the precision of their results, since basically we use *Mathematica*'s own arithmetic for estimations. All the implementations depend on *Mathematica*'s accuracy and precision tracking system, though in our implementation, we round up all the error parts in case our assumption is incorrect.

□ Tolerance Examples

Next we show some examples of the SNAP tolerance mechanism. This loads the package.

```
In[5]:= Needs["SNAP`"]
```

```
This package includes routines which provide
SNAP functionalities. The version of this package
is 0.2.2 and implemented by Kosaku NAGASAKA.
```

This gives a SNAP structure. The output looks like a normal expression.

```
In[6]:= g = SNAP[x^5 + 5.503 x^4 + 9.765 x^3 + 7.647 x^2 + 2.762 x + 0.37725]

Out[6]= 0.37725 + 2.762 x + 7.647 x^2 + 9.765 x^3 + 5.503 x^4 + x^5
```

This gives the `FullForm` of the previous SNAP structure.

```
In[7]:= FullForm[g]

Out[7]//FullForm=
SNAP[Plus[Rational[1509, 4000], Times[Rational[1381, 500], x],
  Times[Rational[7647, 1000], Power[x, 2]],
  Times[Rational[1953, 200], Power[x, 3]],
  Times[Rational[5503, 1000], Power[x, 4]], Power[x, 5]],
 1.537912594467730969706440230810229`14.303315010757448*^-15,
 List[List[x, 5]], AbsolutePolynomial2Norm]
```

The error bound of this SNAP structure, which is a a set of polynomials, is given by `Tolerance`.

```
In[8]:= Tolerance[g]

Out[8]= 1.53791259446773 × 10^-15
```

Each calculation enlarges its error bound.

```
In[9]:= g2 = g * g

Out[9]= (0.37725 + 2.762 x + 7.647 x^2 + 9.765 x^3 + 5.503 x^4 + x^5)^2

In[10]:= Tolerance[g2]

Out[10]= 1.04637512745224 × 10^-13
```

By using `Element`, we can check whether a polynomial is included in the given SNAP structure or not. The error bound of the following SNAP structure is about 1.53791×10^{-15}; hence, we have these results.

```
In[11]:= Element[Normal[g] + 1.5*^-15, g]

Out[11]= True

In[12]:= Element[Normal[g] + 1.6*^-15, g]

Out[12]= False
```

`Normal` gives corresponding representative polynomials in the normal form.

```
In[13]:= Normal[g] // FullForm

Out[13]//FullForm=
Plus[Rational[1509, 4000], Times[Rational[1381, 500], x],
 Times[Rational[7647, 1000], Power[x, 2]],
 Times[Rational[1953, 200], Power[x, 3]],
 Times[Rational[5503, 1000], Power[x, 4]], Power[x, 5]]
```

We can expand the expression of the representative polynomial of a SNAP structure.

```
In[14]:= Expand[g2]

Out[14]= 0.142318 + 2.08393 x + 13.3983 x^2 + 49.6097 x^3 + 116.57 x^4 +
  180.499 x^5 + 185.042 x^6 + 122.768 x^7 + 49.813 x^8 + 11.006 x^9 + x^10
```

Any result of a substitution for a SNAP structure also has a guaranteed accuracy.

```
In[15]:= g3 = g2 /. x → 1

Out[15]= 731.93244306250

In[16]:= Tolerance[g3]

Out[16]= 1.15101264019751 × 10^-12
```

We show the situation discussed in the last paragraph of the Tolerance Mechanisms subsection.

```
In[17]:= g

Out[17]= 0.37725 + 2.762 x + 7.647 x^2 + 9.765 x^3 + 5.503 x^4 + x^5

In[18]:= g - x^5

  SNAP::invalid : Tolerance is larger than the
     representative leading coefficient: 0.` ≤ 1.537912594467731`*^-15.

Out[18]= 0.37725 + 2.762 x + 7.647 x^2 + 9.765 x^3 + 5.503 x^4
```

Because this rewriting can be important for users, this package generates the displayed warning.

■ Other Basic Operations

We implemented other basic operations: a polynomial division (quotient and remainder), reductions (for non-univariate polynomials), and root-finding with error considerations.

Function	Purpose
`NSolve`	finds numerical roots with error considerations.
`PolynomialQuotient`	computes a polynomial quotient as a SNAP structure.
`PolynomialRemainder`	computes a polynomial remainder as a SNAP structure.
`PolynomialReduce`	computes a reduced polynomial as a SNAP structure.
`D`	computes derivatives as SNAP structures.

Table 2. Other basic functions.

□ Polynomial Division (Univariate Case)

Let $f(x)$ and $g(x)$ be the following polynomials of degrees n and m, $m \le n$, with tolerances ϵ_f and ϵ_g, respectively:

$$\begin{aligned} f(x) &= f_n x^n + f_{n-1} x^{n-1} + \cdots + f_1 x + f_0, \ f_i \in \mathbb{C}, \\ g(x) &= g_m x^m + g_{m-1} x^{m-1} + \cdots + g_1 x + g_0, \ g_i \in \mathbb{C}. \end{aligned} \tag{25}$$

Dividing $f(x)$ by $g(x)$ is defined as follows, with polynomials $q(x)$ and $r(x)$:

$$f(x) = q(x)\, g(x) + r(x), \ \deg q = n - m, \ \deg r < m. \tag{26}$$

For SNAP structures, we have to check that the polynomials $\overline{q}(x)$ and $\overline{r}(x)$ for the given $\overline{f}(x)$ and $\overline{g}(x)$ are properly related to $q(x)$ and $r(x)$:

$$\begin{aligned} &\overline{f}(x) = \overline{q}(x)\, \overline{g}(x) + \overline{r}(x), \ \deg \overline{q} = n - m, \\ &\deg \overline{r} < m, \ \overline{f} \in P_*(f, \epsilon_f), \ \overline{g} \in P_*(g, \epsilon_g). \end{aligned} \tag{27}$$

We have the following corollaries [11]. Note the discussion at the end of the Tolerance Mechanisms subsection.

In these corollaries, G is the $(n+1) \times (n+1)$ matrix:

$$G = \begin{pmatrix} g_m & 0 & \cdots & \cdots & 0 & 0 \\ g_{m-1} & g_m & \ddots & \ddots & \vdots & \vdots \\ \vdots & g_{m-1} & \ddots & \ddots & \vdots & \vdots \\ g_0 & \vdots & \ddots & \ddots & 0 & \vdots \\ \vdots & \vdots & \ddots & \ddots & g_m & 0 \\ 0 & \cdots & \cdots & \cdots & g_{m-1} & g_m \end{pmatrix}. \tag{28}$$

Corollary 2. *If the following expression holds,*

$$\sigma_2 \sqrt{n+1}\, \epsilon_g < 1, \ \sigma_2 = \|G^{-1}\|_2, \tag{29}$$

we have

$$\begin{aligned} &\forall\, \overline{f} \in P_2(f, \epsilon_f), \ \forall\, \overline{g} \in P_2(g, \epsilon_g), \ \overline{q} \in P_2(q, \epsilon_q), \ \overline{r} \in P_2(r, \epsilon_r) \\ &\epsilon_q = \sigma_2 \left(\epsilon_f + \sqrt{n+1}\, \epsilon_g \left(\|q\|_2 + \sigma_2 \epsilon_f\right) \Big/ \left(1 - \sigma_2 \sqrt{n+1}\, \epsilon_g\right)\right), \\ &\epsilon_r = \epsilon_f + \sqrt{\left\lceil \frac{n}{2} \right\rceil + 1}\, \left(\|g\|_2 \epsilon_q + \|q\|_2 \epsilon_g + \epsilon_g \epsilon_q\right). \end{aligned} \tag{30}$$

Corollary 3. *If the following expression holds,*

$$\sigma_1 \epsilon_g < 1, \ \sigma_1 = \|G^{-1}\|_1, \tag{31}$$

we have

$$\begin{aligned}
&\forall\ \overline{f} \in P_1(f, \epsilon_f),\ \forall\ \overline{g} \in P_1(g, \epsilon_g),\ \overline{q} \in P_1(q, \epsilon_q),\ \overline{r} \in P_1(r, \epsilon_r),\\
&\overline{q} \in P_1(q, \epsilon_q),\ \overline{r} \in P_1(r, \epsilon_r),\\
&\epsilon_q = \sigma_1(\epsilon_f + \epsilon_g(\|q\|_1 + \sigma_1 \epsilon_f)/(1 - \sigma_1 \epsilon_g)),\\
&\epsilon_r = \epsilon_f + \|g\|_1 \epsilon_q + \|q\|_1 \epsilon_g + \epsilon_g \epsilon_q.
\end{aligned} \tag{32}$$

Corollary 4. *If the following expression holds,*

$$\sigma_\infty (n+1)\epsilon_g < 1,\ \sigma_\infty = \|G^{-1}\|_\infty, \tag{33}$$

we have

$$\begin{aligned}
&\forall\ \overline{f} \in P_\infty(f, \epsilon_f),\ \forall\ \overline{g} \in P_\infty(g, \epsilon_g),\ \overline{q} \in P_\infty(q, \epsilon_q),\ \overline{r} \in P_\infty(r, \epsilon_r),\\
&\epsilon_q = \sigma_\infty(\epsilon_f + (n+1)\epsilon_g(\|q\|_\infty + \sigma_\infty \epsilon_f)/(1 - \sigma_\infty (n+1)\epsilon_g)),\\
&\epsilon_r = \epsilon_f + \left(\left\lceil \frac{n}{2} \right\rceil + 1\right)(\|g\|_\infty \epsilon_q + \|q\|_\infty \epsilon_g + \epsilon_g \epsilon_q).
\end{aligned} \tag{34}$$

□ Polynomial Reduction (More than One Variable Case)

Since the SNAP structure is extended to multivariate polynomials in this version, we introduce polynomial reductions. Because a reduction can be done by two multiplications and one subtraction, we just reduce polynomials by ordinary algorithms using the SNAP structures introduced in the previous section. Note that we assume that the given polynomial basis is a Gröbner basis for any possible combinations of polynomials in the basis, and the result, including tolerance, is dependent on a specified term order.

□ Root Finding with Error Considerations

There are many root-finding methods. Basically, *Mathematica* uses the Jenkins–Traub method. Since those roots found by numerical methods may not be the exact ones, we have to consider numerical errors after calculations. For example, Smith [13] studied error bounds for numerical roots.

However, working with polynomials with errors in their coefficients extends the problem. Terui and Sasaki [14] studied an extended version of Smith's work, by which we can bound errors included in numerical roots of polynomials with errors on their coefficients and for polynomials represented as SNAP structures.

Lemma 3 (Statement in [14]). *For any polynomial* $\overline{f}(x) \in P_*(f, \epsilon)$, *we have*

$$|\zeta_i - \overline{\zeta}_{\pi(i)}| \le n \frac{|f(\zeta_i)| + \epsilon \sum_{j=0}^{n} |\zeta_i|^j}{(|f_n| + \epsilon)\left|\prod_{j=1, \ne i}^{n} (\zeta_i - \zeta_j)\right|}, \tag{35}$$

where n *denotes the degree of* $f(x)$, $\zeta_1, \ldots, \zeta_n$ *and* $\overline{\zeta}_{\pi(1)}, \ldots, \overline{\zeta}_{\pi(n)}$ *are the roots of* $f(x)$ *and* $\overline{f}(x)$, *respectively,* $\pi(i)$ *is a permutation of* $\{1, \ldots, n\}$ *that minimizes the maximum distance between the roots:* $\max_i |\zeta_i - \overline{\zeta}_{\pi(i)}|$, *and* f_n *denotes the leading coefficient of* $f(x)$.

□ Partial Derivation with Error Considerations

Computing the partial derivatives of the given polynomial is also important. We have the following trivial lemma to calculate the derivatives of SNAP structures.

Lemma 4. *For any polynomial* $\overline{f}(u_1, \ldots, u_r) \in P_*(f, \epsilon)$, *we have*

$$\frac{\partial \overline{f}(u_1, \ldots, u_r)}{\partial u_i} \in P_*\left(\frac{\partial f(u_1, \ldots, u_r)}{\partial u_i}, \epsilon \times \deg_{u_i} f\right) \qquad (i = 1, \ldots, r). \quad (36)$$

□ Examples

Here we show some examples of basic operations of the package using the previous polynomial.

```
In[19]:= g
```

```
Out[19]= 0.37725 + 2.762 x + 7.647 x^2 + 9.765 x^3 + 5.503 x^4 + x^5
```

The following examples give all the roots of the given representative polynomial with or without considerations of the error bound. We recommend comparing the following two results of `NSolve` and `System`NSolve`. `NSolve` with a SNAP structure considers all the possible polynomials within the structure; hence, their tolerances are larger than that of `NSolve` without a SNAP structure.

This gives a result with error considerations according to Lemma 3.

```
In[20]:= NSolve[g == 0, x]
```

```
Out[20]= {{x → -3.00007208113}, {x → -0.9989903849}, {x → -0.54121518},
  {x → -0.48136118 - 0.02946454 i}, {x → -0.48136118 + 0.02946454 i}}
```

```
In[21]:= Tolerance /@ (x /. %)
```

```
Out[21]= {1.17359067209883 × 10^-12, 3.5889260050551 × 10^-11,
  6.8881497758683 × 10^-9, 6.9645254663057 × 10^-9, 6.9645254663057 × 10^-9}
```

This gives a result without error considerations.

```
In[22]:= System`NSolve[Normal[g] == 0, x]
```

```
Out[22]= {{x → -3.00007}, {x → -0.99899}, {x → -0.541215},
  {x → -0.481361 - 0.0294645 i}, {x → -0.481361 + 0.0294645 i}}
```

```
In[23]:= Tolerance /@ (x /. %)
```

```
Out[23]= {3.3307491000082 × 10^-16, 1.10910212669417 × 10^-16,
  6.0086955683052 × 10^-17, 5.3541849581150 × 10^-17, 5.3541849581150 × 10^-17}
```

To show division examples, define the following two polynomials.

```
In[24]:= g2 = Expand[g * g]
```

```
Out[24]= 0.142318 + 2.08393 x + 13.3983 x^2 + 49.6097 x^3 + 116.57 x^4 +
  180.499 x^5 + 185.042 x^6 + 122.768 x^7 + 49.813 x^8 + 11.006 x^9 + x^10
```

```
In[25]:= h = x^3 + Random[]
```

```
Out[25]= 0.465024 + x^3
```

This gives the quotient and remainder of `g2` by `h` with error considerations. Note that though `h` is not in the SNAP structure, it is automatically transformed into it and SNAP functions are overloaded because the built-in functions are not compatible with such arguments including a SNAP structure.

```
In[26]:= {{PolynomialQuotient[g2, h, x]}, PolynomialRemainder[g2, h, x]}

Out[26]= {{-34.0593 + 59.6968 x + 157.335 x^2 + 179.924 x^3 + 122.303 x^4 +
     49.813 x^5 + 11.006 x^6 + x^7}, 15.9807 - 25.6765 x - 59.7661 x^2}
```

These commands are the same for univariate polynomials.

```
In[27]:= PolynomialReduce[g2, {h}, {x}]

Out[27]= {{-34.0593 + 59.6968 x + 157.335 x^2 + 179.924 x^3 + 122.303 x^4 +
     49.813 x^5 + 11.006 x^6 + x^7}, 15.9807 - 25.6765 x - 59.7661 x^2}
```

In the next example, `PolynomialRemainder` gives a pseudo-zero number since `g` can divide `g2`. However, these polynomials have their error bounds, and we cannot argue that its remainder is completely zero; hence, the following warning messages are generated.

```
In[28]:= {g3, g4} = {PolynomialQuotient[g2, g, x], PolynomialRemainder[g2, g, x]}

SNAP::invalid : Tolerance is larger than the
   representative leading coefficient: 0.` ≤ 1.0292162141035616`*^-6.

General::stop : Further output of
   SNAP::invalid will be suppressed during this calculation. More...

Out[28]= {0.37725 + 2.762 x + 7.647 x^2 + 9.765 x^3 + 5.503 x^4 + x^5, 0. × 10^-6}
```

SNAP functions give normal *Mathematica* numbers if their degrees are not larger than zero.

```
In[29]:= {SNAPQ[g3], SNAPQ[g4]}

Out[29]= {True, False}
```

Each operation or calculation enlarges error bounds; hence, tolerances of the roots also get larger.

```
In[30]:= NSolve[g3 == 0, x]

Out[30]= {{x → -3.00007}, {x → -0.99899}, {x → -0.5412},
      {x → -0.4814 - 0.0295 i}, {x → -0.4814 + 0.0295 i}}

In[31]:= Tolerance /@ (x /. %)

Out[31]= {1.76394520485190 × 10^-6, 3.67660059293547 × 10^-6,
      0.0000641619768854552, 0.0000561916017665273, 0.0000561916017665273}
```

The tolerance correction mechanism used in `NSolve` with SNAP structures assumes that the given representative polynomial does not have multiple roots. Therefore, the following warning message is generated if the given polynomial has multiple (or close) roots and any tolerance correction is not applied. This means that any output of `Tolerance` below is not reliable.

In[32]:=
```
NSolve[g2 == 0, x]
```

```
SNAP::multipleroots : The given representative polynomial
   has multiple roots. Tolerances of the roots can not be computed.
```

Out[32]=
```
{{x → -3.000072}, {x → -3.000072}, {x → -0.99899},
 {x → -0.99899}, {x → -0.54}, {x → -0.54}, {x → -0.48 - 0.03 i},
 {x → -0.48 + 0.03 i}, {x → -0.48 - 0.03 i}, {x → -0.48 + 0.03 i}}
```

In[33]:=
```
Tolerance /@ (x /. %)
```

Out[33]=
$$\{4.77743842704102\times 10^{-7},\ 4.77743842704538\times 10^{-7},\ 1.63209073103228\times 10^{-6},\ 1.63209073944919\times 10^{-6},\ 0.00376341857130941,\ 0.00376343263094951,\ 0.00233327894319648,\ 0.00233327894319648,\ 0.00233326644754287,\ 0.00233326644754287\}$$

Partial derivatives also can be computed in SNAP structures.

In[34]:=
```
g2d = D[g2, x]
```

Out[34]=
$$2.08393 + 26.7966\,x + 148.829\,x^2 + 466.282\,x^3 + 902.495\,x^4 + 1110.25\,x^5 + 859.373\,x^6 + 398.504\,x^7 + 99.054\,x^8 + 10.\,x^9$$

In[35]:=
```
NSolve[g2d == 0, x]
```

Out[35]=
```
{{x → -3.00007208}, {x → -2.5308273}, {x → -0.9990},
 {x → -0.8692}, {x → -0. × 10^-1}, {x → 0. × 10^1}, {x → 0. × 10^1},
 {x → -0. × 10^-1 + 0. × 10^-1 i}, {x → -0. × 10^-1 + 0. × 10^-1 i}}
```

In[36]:=
```
Tolerance /@ (x /. %)
```

Out[36]=
$$\{8.1322107325183\times 10^{-9},\ 2.16884179842690\times 10^{-8},\ 0.0000106535444881950,\ 0.0000474506566469721,\ 0.4545922961971249,\ 12.83683482913959,\ 10.16786064649990,\ 0.3798417168646968,\ 0.3798417168646968\}$$

The tolerance correcting method used in SNAP computes subtractions among close numbers so it requires a certain precision. In this case, working precision is not enough; hence, we increase it.

In[37]:=
```
NSolve[g2d == 0, x, 32]
```

Out[37]=
```
{{x → -3.000072081}, {x → -2.530827278}, {x → -0.998990385},
 {x → -0.86922548}, {x → -0.541215}, {x → -0.50942}, {x → -0.49292},
 {x → -0.481361 - 0.029465 i}, {x → -0.481361 + 0.029465 i}}
```

In[38]:=
```
Tolerance /@ (x /. %)
```

Out[38]=
$$\{1.42670938968283\times 10^{-10},\ 1.61240156747135\times 10^{-10},\ 7.7305177908929\times 10^{-10},\ 1.75820169776709\times 10^{-9},\ 4.08068757681530\times 10^{-7},\ 2.48951715376255\times 10^{-6},\ 2.38910508764410\times 10^{-6},\ 3.44787932738920\times 10^{-7},\ 3.44787932738920\times 10^{-7}\}$$

■ Symbolic-Numeric Algorithm Implementation

Using the basic features, we have started to modify and implement known symbolic-numeric algorithms. In the current implementation, only one algorithm for each computation is used. Other algorithms will be implemented in the near future.

Function	Purpose
`PolynomialGCD`	computes an ϵ-GCD of the two given polynomials.
`CoprimeQ`	gives `True` if the two polynomials are coprime within the tolerance.
`ApproximateDivisorQ`	gives `True` if the first argument is approximately divisible by the second.
`ApproximateQuotient`	gives a quotient if `ApproximateDivisorQ` is `True`.
`NearestSingularPolynomial`	computes the nearest singular polynomial (tolerance is not considered).
`AbsolutelyIrreducibleQ`	gives `True` if the polynomial is absolutely irreducible within the tolerance.
`SeparationBound`	computes a separation bound (or irreducibility radius) of the given polynomial.
`Factor`	factors the given monic polynomial numerically.
`FactorList`	gives a list of pseudo-factors of the given monic polynomial.

Table 3. Symbolic numeric functions.

□ Approximate GCD and Divisors (Univariate Case)

From the early historical period of symbolic-numeric computations, various approximate GCDs have been studied. The problem treated here is very simple: for the given polynomials $g(x)$ and $h(x)$ and the tolerance ϵ, find a polynomial $f(x)$ of maximal degree that satisfies

$$\exists\, \overline{g}(x) \in P_*(g,\ \epsilon\|g\|_*),\ \exists\, \overline{h}(x) \in P_*(h,\ \epsilon\|h\|_*),\ f(x) \mid \overline{g}(x),\ f(x) \mid \overline{h}(x), \qquad (37)$$

where $*$ denotes 2, 1, or ∞. The polynomial $f(x)$ is called an ϵ-GCD of polynomials $g(x)$ and $h(x)$ with tolerance ϵ. Currently, we have implemented the algorithm by Pan [2], for the 2-norm case only.

We note that there are also other approximate GCDs that have slightly different definitions and approximate GCDs of multivariate polynomials. Those approximate GCDs will be implemented in a future release.

Considering approximate GCDs, the following concept of an ϵ-divisor is useful. For the given polynomial $g(x)$ and the tolerance ϵ, we call a polynomial $f(x)$ an ϵ-divisor of $g(x)$ if $f(x)$ satisfies

$$\exists\, \overline{g}(x) \in P_*(g,\ \epsilon \|g\|_*),\ f(x) \mid \overline{g}(x), \tag{38}$$

where $*$ denotes 2, 1, or ∞. Moreover, in this package, we call the quotient of $\overline{g}(x)$ by $f(x)$ an ϵ-quotient of $g(x)$ by $f(x)$. These concepts are used in Pan's algorithm, and currently we have only implemented the 2-norm case.

Note that, theoretically, ϵ-GCD, ϵ-divisor, and ϵ-quotient are exact polynomials, since only the given polynomials have perturbation parts, and ϵ-GCD, ϵ-divisor, and ϵ-quotient are treated as exact polynomials in those computations. However, due to numerical errors, in this package, ϵ-GCD, ϵ-divisor, and ϵ-quotient are treated as SNAP structures.

We also provide a coprimeness check function that uses the well-known fact that if the Sylvester matrix of the given two polynomials has full rank, then they are coprime [15].

□ Nearest Singular Polynomial (Univariate Case)

The nearest singular polynomial [16, 17, 18] of $f(x)$ is the nearest polynomial $\overline{f}(x)$ that has a double root, minimizes $\|f(x) - \overline{f}(x)\|$, and has the same degree as $f(x)$. A similar problem that finds the nearest polynomial with constrained roots has been studied in [19, 20].

In this package, finding the nearest singular polynomial can be written as follows.

For the given polynomial $f(x)$ and tolerance ϵ, find a polynomial $\overline{f}(x)$ satisfying

$$\overline{f}(x) \in P_*(f,\ \epsilon),\ \exists\, c \in \mathbb{C},\ (x-c)^2 \mid \overline{f}(x), \tag{39}$$

where $*$ denotes 1, 2, or ∞; if the output is `False`, the nearest singular polynomial does not exist in the given SNAP structure. The current version of the package solves this problem using the known algorithm [18], so the command can only solve the problem for the 2-norm case. The constrained roots version of the problem will be solvable in a future release.

Note that the current implementation outputs a normal polynomial (not in a SNAP structure) and the given tolerance for the command corresponding to the nearest singular polynomial does not have the same meaning as the other commands of the *SNAP* package. For more information, see [18].

□ Irreducibility Testing for Bivariate Polynomials

Conventional ordinary factorization algorithms may always output "absolutely irreducible" for numerical or empirical polynomials, since the given polynomial

may have error parts on its coefficients even if the original polynomial is reducible. Moreover, if a numerical factorization algorithm, for example [7], outputs "no nontrivial factors found," it does not mean "absolutely irreducible," since those algorithms can basically find factors when the given polynomial is close enough to a reducible polynomial. Hence, the "irreducibility testing" problem is still important for numerical or empirical polynomials [21, 22, 23, 24, 25].

In this package, the problem becomes:

> For the given SNAP structure $P_*(f, \epsilon)$, prove that any polynomial $\overline{f}(u_1, \ldots, u_r) \in P_*(f, \epsilon)$ is absolutely irreducible.

The algorithm implemented in this package (Nagasaka [23]) is an improved version of the algorithm of Kaltofen and May [22] for bivariate polynomials. Note that the current implementation is based on the algorithm for the 2-norm case; hence, there are possibilities of improvement for another norm. The version for more than two variables will be implemented in a future release. The largest problem in implementing more than two variables is effectiveness, and further studies are needed.

The previously mentioned methods [22, 23] use the coefficients of the given polynomial directly, so we can adapt it to *Mathematica*'s coefficient-wise accuracy concept. This is better than the original methods, because treating tolerances as polynomial norms tends to overestimate. The current implementation can do this for polynomials not in SNAP structures.

□ Numerical Factorization of Multivariate Polynomials

For the same reason as the previous test for irreducibility, we have to use completely different factorization algorithms for numerical or empirical polynomials. In this package, we have implemented Sasaki's algorithm [7] with a degree bound studied by Bostan et al. [26]. Currently, for nonmonic polynomials, the command is not stable since none of the approximate GCD algorithms for multivariate polynomials that are needed for factoring nonmonic polynomials are implemented.

Note that the given polynomial may have approximate factors (or so-called numerical or pseudo-factors) even if the algorithm outputs "absolutely irreducible" or "no factors." Therefore, you are encouraged to use the preceding irreducibility testing when you do not get approximate factors.

□ Examples

Here we show some examples of SNAP operations using this previously defined polynomial.

```
In[39]:= g
Out[39]= 0.37725 + 2.762 x + 7.647 x^2 + 9.765 x^3 + 5.503 x^4 + x^5
```

Here we introduce another polynomial. Though it is not in a SNAP structure, treating it as such is acceptable.

```
In[40]:= h = 1.3883 + 4.417 x + 3.8861 x^2 + 0.85593 x^3
Out[40]= 1.3883 + 4.417 x + 3.8861 x^2 + 0.85593 x^3
```

The built-in function outputs that these polynomials are coprime.

```
In[41]:= System`PolynomialGCD[Normal[g], h]
Out[41]= 1/4000
```

The *SNAP* package can compute the ϵ-GCD as follows, where $\epsilon = 0.0001$. Note that the current implementation of `PolynomialGCD` for SNAP structures is still experimental and that the definition of the greatest common divisor of SNAP structures may change in the future.

```
In[42]:= gcdofgh = PolynomialGCD[g, h, 0.0001]
SNAP::preliminary : Preliminary implemented function is called.
Out[42]= 0.00005 + 0.000159079 x + 0.000139959 x^2 + 0.0000308266 x^3
```

We can check its approximate divisibility.

```
In[43]:= ApproximateDivisorQ[g, gcdofgh, 0.0001]
Out[43]= True
```

`ApproximateQuotient` minimizes $\|g - \text{gcdofgh} \times \text{aqofgh}\|_2$ so `aqofgh` is not a constant in this case.

```
In[44]:= aqofgh = ApproximateQuotient[g, gcdofgh, 0.0001]
Out[44]= 7543.97 + 31237.7 x + 32439. x^2
```

Without any tolerance, the worst tolerance between the given polynomials is used.

```
In[45]:= PolynomialGCD[g, h] // Timing
SNAP::toleranceadjusted :
 The given different tolerances adjusted into their maximum.
SNAP::preliminary : Preliminary implemented function is called.
Out[45]= {0.14 Second, 1}
```

This also checks whether they are coprime or not within their tolerance.

```
In[46]:= CoprimeQ[g, h] // Timing
```

```
SNAP::toleranceadjusted :
 The given different tolerances adjusted into their maximum.
```

```
SNAP::machineprecision :
 SNAP function encounters a machine precision number. Machine
   precision numbers may not have enough accuracy
   due to Mathematica's inner operation policy.
```

```
Out[46]= {0.01 Second, True}
```

This gives the nearest singular polynomial to g, so the output polynomial nsg has a double root. Note that the current implementation of NearestSingular⁻Polynomial is still experimental and that the definition of the nearest singular polynomial of a SNAP structure may change in the future.

```
In[47]:= nsg = NearestSingularPolynomial[g]
```

```
SNAP::preliminary : Preliminary implemented function is called.
```

```
Out[47]= 0.377203519748381370167757890889839681876360042214065672119776 97+
 2.762022941524044433428350489406395857621869366526472785623137 79x +
 7.646988676620561357622504161731099465008522214447595797886506014
  x^2 +
 9.765005588945253294112702930232286019423905685965456360726987501
  x^3 +
 5.502997241432276152366822036470183607747161460093308123512301501
  x^4 + x^5
```

```
In[48]:= x /. NSolve[nsg == 0, x]
```

```
SNAP::multipleroots : The given representative polynomial
   has multiple roots. Tolerances of the roots can not be computed.
```

```
Out[48]= {-3.000053130650945683752304343186078410573974339841422837810588,
 -0.999351281713466058303241044864426013564255261286856525001 66,
 -0.516441372459217509897207664836573546919334279320842452221,
 -0.493575728304323450207034491 79, -0.493575728304323450207034491 79}
```

To show an example of absolute irreducibility testing, we define the following bivariate polynomial.

```
In[49]:= f = Expand[(x^2 + y * x + 2 y - 1) (x^3 + y^2 x - y + 7) + 0.2 x]
```

$$\text{Out[49]}= -7 + 0.2\,x + 7\,x^2 - x^3 + x^5 + 15\,y + 7\,x\,y - x^2\,y + 2\,x^3\,y + x^4\,y - 2\,y^2 - 2\,x\,y^2 + x^3\,y^2 + 2\,x\,y^3 + x^2\,y^3$$

This tests its absolute irreducibility within the given tolerance 0.00001.

```
In[50]:= AbsolutelyIrreducibleQ[f, 0.00001]
```

```
SNAP::machineprecision :
 SNAP function encounters a machine precision number. Machine
   precision numbers may not have enough accuracy
   due to Mathematica's inner operation policy.
```

```
Out[50]= True
```

If the given polynomial has a SNAP structure, its tolerance is used, so the following evaluations give the same result.

```
In[51]:= fs = SNAP[f, 0.00001]
```

$$\text{Out[51]=}\quad -7.+0.2\,x+7.\,x^2-1.\,x^3+x^5+15.\,y+7.\,x\,y-1.\,x^2\,y+2.\,x^3\,y+x^4\,y-2.\,y^2-2.\,x\,y^2+x^3\,y^2+2.\,x\,y^3+x^2\,y^3$$

```
In[52]:= AbsolutelyIrreducibleQ[fs]
```

```
Out[52]= True
```

We can also compute a separation bound of `f`. In this case, all the polynomials of $P_2(f, 0.000791622)$ are absolutely irreducible.

```
In[53]:= SeparationBound[f]
```

```
SNAP::machineprecision :
 SNAP function encounters a machine precision number. Machine
   precision numbers may not have enough accuracy
   due to Mathematica's inner operation policy.
```

```
Out[53]= 0.000791622
```

A warning message is generated if the package routines encounter a machine precision number. We recommend not using machine precision numbers.

```
In[54]:= f = Expand[(x^2 + y * x + 2 y - 1) (x^3 + y^2 x - y + 7) + 0.2``16 x]
```

$$\text{Out[54]=}\quad -7+0.200000000000000\,x+7\,x^2-x^3+x^5+15\,y+7\,x\,y-x^2\,y+2\,x^3\,y+x^4\,y-2\,y^2-2\,x\,y^2+x^3\,y^2+2\,x\,y^3+x^2\,y^3$$

```
In[55]:= SeparationBound[f]
```

```
Out[55]= 0.0007916215679
```

This gives an example using the algorithm adapted for *Mathematica*'s coefficient-wise accuracy concept. Hence, changing all the coefficients within their accuracy does not change its absolute irreducibility.

```
In[56]:= AbsolutelyIrreducibleQ[f]
```

```
Out[56]= True
```

For numerical polynomials, the built-in function gives the input.

```
In[57]:= f2 = SetPrecision[Expand[(x^2 + y * x + 2 y - 1) (x^3 + y^2 x - y + 7)], 16]

Out[57]= -7.000000000000000 + 7.000000000000000 x^2 - 1.000000000000000 x^3 + x^5 +
  15.00000000000000 y + 7.000000000000000 x y - 1.000000000000000 x^2 y +
  2.000000000000000 x^3 y + x^4 y - 2.000000000000000 y^2 -
  2.000000000000000 x y^2 + x^3 y^2 + 2.000000000000000 x y^3 + x^2 y^3

In[58]:= System`Factor[f2]

Out[58]= 1.000000000000000
  (-7.00000000000000 + 7.00000000000000 x^2 - 1.000000000000000 x^3 +
    1.000000000000000 x^5 + 15.00000000000000 y + 7.00000000000000 x y -
    1.000000000000000 x^2 y + 2.000000000000000 x^3 y +
    1.000000000000000 x^4 y - 2.000000000000000 y^2 -
    2.000000000000000 x y^2 + 1.000000000000000 x^3 y^2 +
    2.000000000000000 x y^3 + 1.000000000000000 x^2 y^3)
```

With the *SNAP* package, for example, we can factor it numerically with a backward error bound.

```
In[59]:= Chop@Factor[f2, 0.0000001]

Out[59]= (-1.000000000000000 + x^2 + (2.0000000000000 + 1.0000000000000 x) y)
  (7.00000000000000 + x^3 - 1.0000000000000 y + 1.000000000000 x y^2)
```

If we use a SNAP structure, its tolerance is automatically used as a backward error bound.

```
In[60]:= f2s = SNAP[Expand[(x^2 + y * x + 2 y - 1) (x^3 + y^2 x - y + 7)], 0.0000001]

Out[60]= -7. + 7. x^2 - 1. x^3 + x^5 + 15. y + 7. x y - 1. x^2 y +
  2. x^3 y + x^4 y - 2. y^2 - 2. x y^2 + x^3 y^2 + 2. x y^3 + x^2 y^3

In[61]:= Chop@Factor[f2s]

Out[61]= (-1.000000000000000 + x^2 + (2.0000000000000 + 1.0000000000000 x) y)
  (7.00000000000000 + x^3 - 1.0000000000000 y + 1.000000000000 x y^2)
```

■ Conclusion

The *SNAP* package is useful for almost all users who have to work with polynomials with errors in their coefficients. Users may think that *Mathematica* has its own accuracy and precision system, and therefore another structure like those in SNAP is unnecessary. This will be true in the future; however, at least now, most of the latest algorithms for numerical or empirical polynomials cannot operate with coefficient-wise accuracy and precision. Using only significant digits like *Mathematica*'s cannot answer the algebraic problems, though it can guarantee significant digits of coefficients generated by polynomial arithmetic. Moreover, most of the algorithms in symbolic-numeric computations have to use matrix computations and are not compatible with coefficient-wise concepts, since they usually use matrix norms. Depending on the algorithm, by using absolute irreducibility testing, for example, we can combine them with *Mathematica*'s coefficient-wise error scheme and we plan to incorporate that in a future release.

Moreover, we are considering whether computing Canonical Comprehensive Gröbner Bases (CCGB) should be integrated into the *SNAP* package since we have implemented CCGB in *Mathematica* and some kind of CCGB is the only way to treat numerical errors exactly. They can be represented as parameters on coefficients; however, this method is so time-consuming that this release does not have CCGB routines. We note that *Mathematica* can compute Gröbner bases numerically, but we think any result is not guaranteed mathematically. However, *Mathematica*'s built-in computation of numerical Gröbner bases is more advanced than finding pseudo-solutions numerically, which is very difficult, and there are few known academic results.

■ Acknowledgment

This research was partially supported by Grants-in-Aid for Scientific Research from the Japanese Ministry of Education, Culture, Sports, Science and Technology (#16700016). We also wish to thank the anonymous referees for their useful suggestions.

■ References

[1] V. Y. Pan, "Approximate Polynomial GCDs, Padé Approximation, Polynomial Zeros, and Bipartite Graphs," in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, New York: ACM Press, and Philadelphia: SIAM Publications, 1998 pp. 68–77.

[2] V. Y. Pan, "Computation of Approximate Polynomial GCDs and an Extension," *Information and Computation*, **167**(2), 2001 pp. 71–85.

[3] B. Beckermann and G. Labahn, "When Are Two Numerical Polynomials Relatively Prime?" *Journal of Symbolic Computation*, **26**(6), 1998 pp. 677–689.

[4] I. Z. Emiris, A. Galligo, and H. Lombardi, "Certified Approximate Univariate GCDs," *Journal of Pure and Applied Algebra (Special Issue on Algorithms in Algebra)*, **117** & **118**, 1997 pp. 229–251.

[5] R. M. Corless, M. W. Giesbrecht, M. van Hoeij, I. S. Kotsireas, and S. M. Watt, "Towards Factoring Bivariate Approximate Polynomials," in *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation (ISSAC 2001)*, London, Ontario, Canada, New York: ACM Press, 2001 pp. 85–92.

[6] Z. Mou-tan and R. Unbehauen, "Approximate Factorization of Multivariate Polynomials," *Signal Processing*, **14**, 1988 pp. 141–152.

[7] T. Sasaki, "Approximate Multivariate Polynomial Factorization Based on Zero-Sum Relations," in *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation (ISSAC 2001)*, London, Ontario, Canada, New York: ACM Press, 2001 pp. 284–291.

[8] Y. Huang, W. Wu, H. J. Stetter, and L. Zhi, "Pseudofactors of Multivariate Polynomials," in *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation (ISSAC 2000)*, St. Andrews, UK, New York: ACM Press, 2000 pp. 161–168.

[9] F. Kako and T. Sasaki, "Proposal of 'Effective Floating-Point Number' for Approximate Algebraic Computation," *ACM SIGSAM Bulletin,* **31**(3), 1997 p. 31.

[10] K. Nagasaka, "*SNAP* Package," (Talk in Japanese), *JSSAC 2004*, 2004.

[11] K. Nagasaka, "*SNAP* Package for *Mathematica* and Its Applications," in *The Ninth Asian Technology Conference in Mathematics (ATCM 2004)*, Singapore, 2004.

[12] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed., Johns Hopkins Studies in Mathematical Sciences, Baltimore: The Johns Hopkins University Press, 1996.

[13] B. T. Smith, "Error Bounds for Zeros of a Polynomial Based upon Gerschgorin's Theorems," *Journal of the ACM (JACM)*, **17**(4), 1970 pp. 661–674.

[14] A. Terui and T. Sasaki, "'Approximate Zero-Points' of Real Univariate Polynomial with Large Error Terms," *Journal (Information Processing Society of Japan)*, **41**(4), 2000 pp. 974–989.

[15] Z. Zeng, "The Approximate GCD of Inexact Polynomials. Part I: A Univariate Algorithm," Preprint, 2004.

[16] N. Karmarkar and Y. N. Lakshman, "Approximate Polynomial Greatest Common Divisors and Nearest Singular Polynomials," in *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation (ISSAC 1996)*, Zurich, Switzerland, New York: ACM Press, 1996 pp. 35–39.

[17] L. Zhi and W. Wu, "Nearest Singular Polynomials," *Journal of Symbolic Computation*, **26**(6), 1998 pp. 667–675.

[18] L. Zhi, W. Wu, M.-T. Noda, and H. Kai, "Hybrid Method for Computing the Nearest Singular Polynomials," *MM Research Preprints*, **20**, 2001 pp. 229–239.

[19] M. A. Hitz and E. Kaltofen, "Efficient Algorithms for Computing the Nearest Polynomial with Constrained Roots," in *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation (ISSAC 1998)*, Rostock, Germany, New York: ACM Press, 1998 pp. 236–243.

[20] M. A. Hitz, E. Kaltofen, and Y. N. Lakshman, "Efficient Algorithms for Computing the Nearest Polynomial with a Real Root and Related Problems," in *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation (ISSAC 1999)*, Vancouver, B.C., Canada, New York: ACM Press, 1999 pp. 205–212.

[21] E. Kaltofen, "Effective Noether Irreducibility Forms and Applications," *Journal of Computer and System Sciences*, **50**(2), 1995 pp. 274–295.

[22] E. Kaltofen and J. May, "On Approximate Irreducibility of Polynomials in Several Variables," in *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC 2003)*, Philadelphia, PA, New York: ACM Press, 2003 pp. 161–168.

[23] K. Nagasaka, "Towards More Accurate Separation Bounds of Empirical Polynomials," *ACM SIGSAM Bulletin (Formally Reviewed Articles)*, **38**(4), 2004 pp. 119–129.

[24] K. Nagasaka, "Neighborhood Irreducibility Testing of Multivariate Polynomials," in *Proceedings of the Sixth International Workshop on Computer Algebra in Scientific Computing (CASC 2003)*, Passau, Germany, New York: ACM Press, 2003 pp. 283–292.

[25] K. Nagasaka, "Towards Certified Irreducibility Testing of Bivariate Approximate Polynomials," in *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC 2002)*, Lille, France, New York: ACM Press, 2002 pp. 192–199.

[26] A. Bostan, G. Lecerf, B. Salvy, É. Schost, and B. Wiebelt, "Complexity Issues in Bivariate Polynomial Factorization," in *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC 2004)*, Santander, Spain, New York: ACM Press, 2004 pp. 42–49.

[27] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt, "The Singular Value Decomposition for Polynomial Systems," in *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC 1995)*, Montreal, Canada, New York: ACM Press, 1995 pp. 195–207.

[28] S. Gao and V. M. Rodrigues, "Irreducibility of Polynomials Modulo *p* via Newton Polytopes," *Journal of Number Theory*, **101**, 2003 pp. 32–47.

[29] W. M. Ruppert, "Reducibility of Polynomials $f(x, y)$ Modulo p," *Journal of Number Theory*, **77**, 1999 pp. 62–70.

[30] H. J. Stetter, *Numerical Polynomial Algebra*, Philadelphia: SIAM, 2004.

[31] K. Nagasaka, "Towards More Accurate Separation Bounds of Empirical Polynomials II," in *Proceedings of the Eighth International Workshop on Computer Algebra in Scientific Computing (CASC 2005)*, Kalamata, Greece, *Lecture Notes in Computer Science*, **3718**, New York: Springer-Verlag, 2005 pp. 318–329.

[32] K. Nagasaka, "Using Coefficient-Wise Tolerance in Symbolic-Numeric Algorithms for Polynomials," *Sushikisyori*, **12**(3), 2006 pp. 21–30.

K. Nagasaka, "Symbolic-Numeric Algebra for Polynomials," *The Mathematica Journal*, 2012. dx.doi.org/10.3888/tmj.10.3-10.

About the Author

Kosaku Nagasaka is an assistant professor at Kobe University in Japan. In the summer of 1999, Nagasaka participated in the Wolfram Research student internship program. Since 2001, he has been one of the directors of the Japanese *Mathematica* User Group. His main research topic is symbolic numeric algorithms for polynomials.

Kosaku Nagasaka
Division of Mathematics and Informatics
Department of Science of Human Environment
Faculty of Human Development
Kobe University
Japan
nagasaka@main.h.kobe-u.ac.jp
wwwmain.h.kobe-u.ac.jp/~nagasaka/research/snap/index.phtml.en