SNAP User's Guide

Claude-Pierre Jeannerod
LIP, Ecole Normale Supérieure de Lyon,
46 Allée d'Italie, 69364 Lyon Cedex 07, France
e–mail: Claude-Pierre.Jeannerod@ens-lyon.fr
and
George Labahn
Department of Computing Science
University of Waterloo, Waterloo, Ontario, Canada
e–mail: glabahn@scg.math.uwaterloo.ca

1 Introduction

There are a number of application areas which can be formally described in terms of algebraic operations on polynomials and matrix polynomials such as division, remainders and greatest common divisors. Areas which make use of such an approach include control theory, linear systems theory and signal processing [15]. While an algebraic polynomial formalism is extremely useful for theoretical studies of properties, it has drawbacks when one tries to take advantage for computations. The primary problem is that many applications use inexact data and finite precision computations while the formalism assumes exact information and error-free computations [17, 21].

In this paper we describe the SNAP (Symbolic-Numeric Algorithms for Polynomials) package for computing with polynomials having inexact coefficients. This package is a first attempt to provide the standard functionalities for inexact polynomials that exist for exact polynomials, including the taking of quotients and remainders, determining if two polynomials are relatively prime and finding greatest common divisors (GCDs). The package is included in the coming release of the MAPLE computer algebra system.

Several algorithms exist for performing such operations, in particular for computing the various notions of the so-called approximate GCDs. These algorithms include modifications of the Euclidean algorithm [11, 17, 20], optimization techniques [6, 13, 14], subresultant-based algorithms [7, 8], and polynomial root approximation [18]. For the most part, the previously mentioned algorithms resort to either infinite or adaptive precision. In our case, the software we present works under the customary model of numerical computation using fixed precision floating-point arithmetic and the underlying algorithms benefit from round-off error analysis which ensures numerical stability [10].

This approach, based on estimating the distance to the closest pair of polynomials having a common factor [3], also allows for computations that typically run in quadratic time. Also, the two approximate GCDs that we focus on in the package (quasi-GCD [20] and ϵ-GCD [3]) are certified in the sense of Emiris *et al* [8].

The remainder of the paper is organized as follows. In the next section, we discuss the numerical computation of quotients and remainders. Section 3 is devoted to testing numerical coprimeness: Section 3.1 recalls the algorithm COPRIME [3], Section 3.2 details how the intermediate linear systems can be solved in a fast and stable way and Section 3.3 deals with the particular case of equal degree polynomials. Section 4 is devoted to the possibility of getting approximate GCDs from the output of the algorithm COPRIME. In Section 5, implementation aspects of the SNAP package are considered, together with some timings, test examples and a description of each functionality of the package.

For the remaining part of the paper, $a(z) = \sum_{i=0}^{m} a_i z^i$ and $b(z) = \sum_{i=0}^{m} b_i z^i$ are univariate complex polynomials with respective degrees m and n. We denote by $\underline{a}(z)$ and $\underline{b}(z)$ their "mirror" counterparts $\underline{a}(z) = z^m a(z^{-1})$ and $\underline{b}(z) = z^n b(z^{-1})$.

2 Quotient and remainder

Our first goal is to compute in a numerically stable manner the coefficients of the polynomial pair (q, r) such that $a(z) = b(z)q(z) + r(z)$ and $\deg r < \deg b$. When $m < n$ we can clearly take $(q, r) = (0, a)$. On the other hand, when $m \geq n$, we see that determining the quotient $q(z) = \sum_{i=0}^{m-n} q_i z^i$ is equivalent to solving the Toeplitz linear system of order $m - n + 1$

$$\begin{bmatrix} b_n & & \\ \vdots & \ddots & \\ b_{2n-m} & \cdots & b_n \end{bmatrix} \begin{bmatrix} q_{m-n} \\ \vdots \\ q_0 \end{bmatrix} = \begin{bmatrix} a_m \\ \vdots \\ a_n \end{bmatrix}. \tag{1}$$

Here and hereafter the ith coefficient of a polynomial is zero whenever $i < 0$ and the blanck areas in matrix representation are assumed to be filled with zero entries.

After obtaining a backward stable solution to (1), we deduce the remainder $r(z)$ from the identity $r(z) = a(z) - b(z)q(z)$.

3 Primality testing

The remaining functions in the current version of SNAP are all based on the following approach, developed by Beckermann and Labahn [2, 3]: compute polynomial solutions $u(z), v(z), \underline{u}(z), \underline{v}(z)$ to the Diophantine equations

$$\begin{aligned} a(z)v(z) + b(z)u(z) &= 1, \quad \deg u < m, \quad \deg v < n, & (2)\\ a(z)\underline{v}(z) + b(z)\underline{u}(z) &= z^{m+n-1}, \quad \deg \underline{u} < m, \quad \deg \underline{v} < n, & (3) \end{aligned}$$

in a fast and numerically stable way. Here, "fast" means arithmetic complexity in $O((m+n)^2)$; "stable" means weakly stable in the sense of Bunch [4]: the forward error is small when the problem is well-conditioned.

This approach provides a numerical coprimeness test together with Bézout coefficients. Additionally, a solution $(u, v, \underline{u}, \underline{v})$ to Equations (2), (3) yields a sharp estimation of the distance $\epsilon(a, b)$ to the set of polynomial pairs with a common root. Indeed, denoting by $|| \; ||$ the 1-norm for the space of matrix polynomials over $\mathbb{C}$, one has

$$\epsilon(a, b) = \min\{||(a - a^*, b - b^*)|| \; : \; \deg \gcd(a^*, b^*) > 0, \deg a^* \leq m, \deg b^* \leq n\}. \tag{4}$$

If we let $S(a, b)$ be the Sylvester matrix associated with $a(z)$, $b(z)$ then it is shown in [2] that

$$\frac{1}{||S(a,b)^{-1}||} \leq \frac{1}{\kappa} \leq \epsilon(a, b) \quad \text{where} \quad \kappa = \left|\left|\begin{bmatrix} v & \underline{v} \\ u & \underline{u} \end{bmatrix}\right|\right|. \tag{5}$$

This improves upon the well known lower bound $1/||S(a, b)^{-1}||$ for $\epsilon(a, b)$.

Of course, a solution to Equations (2), (3) may not exist. However, we will recall that the COPRIME algorithm of [3] always returns a reduced pair of polynomials from which one often can deduce (under additional assumptions) some approximate GCDs.

We restrict to the case where $m > n$ in sections 3.1 and 3.2. The case where $m \leq n$ is considered in section 3.3.

3.1 The algorithm COPRIME

We may assume w.l.o.g. that the input numerical polynomials have been scaled to satisfy $1/2 \leq ||(a,b)|| \leq 1$. Starting with $(a^{(0)}, b^{(0)}, c^{(0)}) = (a(z), b(z), -z^{n-1})$, the COPRIME algorithm computes for $k \leq m$ a numerical polynomial remainder sequence $(a^{(k)}, b^{(k)}, c^{(k)})$ by means of so-called **unimodular reductions** $U^{(k)}(z) \in \mathbb{C}[z]^{2\times 2}$ **of order** k together with **associated vectors** $\underline{U}^{(k)}(z) \in \mathbb{C}[z]^{2\times 1}$. More precisely,

$$(a^{(k)}, b^{(k)}) = (a, b) \cdot U^{(k)}, \quad c^{(k)} = (a, b) \cdot \underline{U}^{(k)} - z^{n+k-1} \tag{6}$$

with $U^{(k)}$ unimodular and where, for $k \geq 1$,

$$n \geq \deg a^{(k)} = m - k > \deg b^{(k)}, \quad \deg c^{(k)} < m - k - 1,$$

$$\deg U^{(k)} \leq \begin{bmatrix} n-m+k-1 & n-m+k \\ k-1 & k \end{bmatrix}, \quad \deg \underline{U}^{(k)} \leq \begin{bmatrix} n-m+k-1 \\ k-1 \end{bmatrix}.$$

The two main features of this Euclidean-like reduction process are summarized in the theorem below.

Theorem 3.1 *The pairs (a, b) and $(a^{(k)}, b^{(k)})$ define two different bases of the same ideal with $a^{(k)}$ of smaller degree than a. When $k = m$, a solution $(u, v, \underline{u}, \underline{v})$ to Equations (2), (3) is given by*

$$\begin{bmatrix} v \\ u \end{bmatrix} = \frac{1}{a^{(m)}(0)} U^{(m)} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} \underline{v} \\ \underline{u} \end{bmatrix} = \underline{U}^{(m)}. \tag{7}$$

$\square$

In finite precision arithmetic, some remainders defined by (6) may correspond to ill-conditioned subproblems in the sense that some Trudy submatrices of $S(a, b)$ may have a large condition number [3, Appendix A]. In order to ensure numerical stability, such remainders are discarded by a "look-ahead" strategy: one jumps from $(a^{(k)}, b^{(k)}, c^{(k)})$ to the first $(a^{(k+s)}, b^{(k+s)}, c^{(k+s)})$ with $s \geq 1$ such that

$$|\det U^{(k+s)}(0)| > \tau \quad \text{and} \quad ||\underline{U}^{(k+s)}|| < 1/\tau. \tag{8}$$

Here, τ is a threshold parameter of order the cubic root of the machine precision and $(a^{(k+s)}, b^{(k+s)})$ is the first basis after $(a^{(k)}, b^{(k)})$ that is **numerically well-behaved** in the sense of (8). The set $\mathcal{A}$ of the indices of all such accepted bases clearly satisfies $\mathcal{A} \subset \{0, 1, 2, \ldots, m\}$.

On the other hand, $U^{(k+s)}$ and $\underline{U}^{(k+s)}$ are determined from $a^{(k)}$, $b^{(k)}$, $c^{(k)}$, $U^{(k)}$, $\underline{U}^{(k)}$ in a numerically stable manner as follows. Consider the $2s \times 2s$ complex matrix

$$M_s^{(k)} = \left[\begin{array}{cccc|cccc} a_{m-k}^{(k)} & & & & 0 & & & \\ a_{m-k-1}^{(k)} & a_{m-k}^{(k)} & & & b_{m-k-1}^{(k)} & \ddots & & \\ \vdots & & \ddots & & \vdots & \ddots & 0 & \\ \vdots & & & a_{m-k}^{(k)} & \vdots & & b_{m-k-1}^{(k)} & \\ \vdots & & & \vdots & \vdots & & \vdots & \\ a_{m-k-2s+1}^{(k)} & \cdots & \cdots & a_{m-k-s}^{(k)} & b_{m-k-2s+1}^{(k)} & \cdots & b_{m-k-s}^{(k)} & \end{array}\right] \tag{9}$$

and solve the three linear systems $M_s^{(k)} x_i = y_i$ whose right hand side is given by

$$y_1 = (0, \ldots, 0, 1)^T, \quad y_2 = -\big(b_{m-k-j}^{(k)}\big)_{j=1,..,2s}, \quad y_3 = \big(c_{m-k-j}^{(k)}\big)_{j=1,..,2s}. \tag{10}$$

We use QR decomposition for solving such systems.

Setting up the 2×2 and 2×1 polynomial matrices

$$U^{(k,k+s)}(z) = \begin{bmatrix} z^{s-1} & \cdots & z^1 & z^0 & 0 & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & z^{s-1} & \cdots & z^1 & z^0 \end{bmatrix} \cdot \Big(x_1, x_2\Big) + \begin{bmatrix} 0 & 0 \\ 0 & z^s \end{bmatrix}$$

and

$$\underline{U}^{(k,k+s)}(z) = \begin{bmatrix} z^{s-1} & \cdots & z^1 & z^0 & 0 & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & z^{s-1} & \cdots & z^1 & z^0 \end{bmatrix} \cdot x_3,$$

one finally obtains[3]

$$U^{(k+s)} = U^{(k)} \cdot U^{(k,k+s)} \quad \text{and} \quad \underline{U}^{(k+s)} = z^s \underline{U}^{(k)} - U^{(k)} \cdot \underline{U}^{(k,k+s)}.$$

When $k + s \in \mathcal{A}$, additional scaling ensures that $1/2 \leq ||U^{(k+s)}|| \leq 1$. The resulting algorithm, called COPRIME, is given in Table 1.

Theorem 3.2 *The algorithm COPRIME is weakly stable and requires in most cases $O((m+n)^2)$ flops.* □

Quadratic complexity can be achieved mainly because for most dense polynomial pairs (a, b) only small jumps ($s \leq 3$) and thus only small linear systems are typically encountered. Note further that the products of (6) can be replaced with the "shorter" products

$$(a^{(k+s)}, b^{(k+s)}) = (a^{(k)}, b^{(k)}) \cdot U^{(k,k+s)}, \quad c^{(k+s)} = z^s c^{(k)} - (a^{(k)}, b^{(k)}) \cdot \underline{U}^{(k,k+s)}.$$

Remark 3.1 *It follows from*

$$S(a,b) = \left[\begin{array}{ccc|ccc} a_m & & & b_n & & \\ \vdots & \ddots & & \vdots & \ddots & \\ a_0 & & a_m & b_0 & & b_n \\ & \ddots & \vdots & & \ddots & \vdots \\ & & a_0 & & & b_0 \end{array}\right] \in \mathbb{C}^{(n+m)\times(n+m)}$$

Method:	Construct (scaled) unimodular reductions $U^{(k)}$ of (a,b) of order $k \in \mathcal{A}$ together with associated vectors $\underline{U}^{(k)}$ for $1 \leq k \leq m$.
Input:	Two polynomials a, b with $\deg a = m > \deg b$. A stability parameter τ.
Output:	If $m \in \mathcal{A}$: RETURN $1/\kappa$ given by (5) and (7). If $m \notin \mathcal{A}$: message since $1/\kappa$ does not exist or is "too small".
Initialization:	$k = 0$, $\mathcal{A} = \{\}$, $U^{(0)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\underline{U}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.
Single Step:	For $s = 1, 2, \ldots$:
	Compute $U^{(k,k+s)}$, $\underline{U}^{(k,k+s)}$ by solving the linear systems (9-10) (if $\det M_s^{(k)} = 0$ then increase s and restart). Rescale $U^{(k)} \cdot U^{(k,k+s)}$ to obtain $U^{(k+s)}$.
Exit s–loop:	if (8) holds. In this case $k \leftarrow k + s$ and $\mathcal{A} \leftarrow \mathcal{A} \cup \{k\}$.
Exit ALGO:	If $k + s = m$.

Table 1: The algorithm COPRIME

that the vectors $[v_{n-1}, \ldots, v_0, u_{m-1}, \ldots, u_0]^T$ *and* $[\underline{v}_{n-1}, \ldots, \underline{v}_0, \underline{u}_{m-1}, \ldots, \underline{u}_0]^T$ *associated with the columns of the matrix polynomial in* κ *are respectively the last and first columns of* $S(a,b)^{-1}$*. On the other hand,* $S(a,b)$ *is a quasi-Toeplitz matrix in the sense that* $S(a,b) - ZS(a,b)Z^T$ *has rank at lost* 2 *with*

$$Z = \begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix} \in \mathbb{C}^{(n+m)\times(n+m)}.$$

Hence it ispossible to solve the two linear systems

$$S(a,b)x = (1, 0, \ldots, 0)^T \quad \textit{and} \quad S(a,b)x = (0, \ldots, 0, 1)^T \tag{11}$$

in a fast and backward stable manner with the algorithm of Chandrasekaran and Sayed [5]. Associated Bézout coefficients obviously follow from this approach; however, it does not allow us to compute approximate GCDs. □

When $m \in \mathcal{A}$ we conclude that (a,b) are numerically coprime up to perturbations of order $\epsilon < 1/\kappa$. When $m \notin \mathcal{A}$ then the last accepted basis $(a^{(k)}, b^{(k)})$ may still provide various approximate GCDs.

3.2 Fast and stable update of QR factorizations

It may happen that the look-ahead strategy yields several jumps of order $s = O(m)$. In this case, successively computing QR decompositions of $M_1^{(k)}, \ldots, M_s^{(k)}$ by the classical method costs $\sum_{i=1}^{s} i^3$

flops, resulting in an $O((m+n)^4)$ algorithm. However, by taking advantage of the recursive structure of the matrix $M_s^{(k)}$ of (9) it is possible to reduce the above complexity to $O((m+n)^3)$. Indeed, we can determine the ith QR decomposition from the $(i-1)$th one in quadratic rather than cubic time. This is precisely why such intermediate linear systems are solved via QR decomposition rather than say gaussian elimination [1].

To see how QR decomposition works with our linear system, we partition the matrix of (9) as $M_s^{(k)} = \left[\, A_s^{(k)} \;\middle|\; B_s^{(k)} \,\right]$ with $A_s^{(k)}$ and $B_s^{(k)}$ of dimensions $2s \times s$. Then one has

$$
M_s^{(k)} = \left[\begin{array}{c|c|c|c}
A_{s-1}^{(k)} & u(a) & B_{s-1}^{(k)} & u(b) \\
& & & \\
\hline
v(a) & a_{m-k-s+1}^{(k)} & v(b) & b_{m-k-s+1}^{(k)} \\
w(a) & a_{m-k-s}^{(k)} & w(b) & b_{m-k-s}^{(k)}
\end{array}\right]
$$

with

$$
\begin{aligned}
u(a) &= (0,\ldots,0,a_{m-k}^{(k)},\ldots,a_{m-k-s+2}^{(k)})^T, \\
u(b) &= (0,\ldots,0,b_{m-k-1}^{(k)},\ldots,b_{m-k-s+2}^{(k)})^T, \\
v(x) &= (x_{m-k-2s+2}^{(k)},\ldots,x_{m-k-s}^{(k)}) \quad \text{for } x \in \{a,b\}, \\
w(x) &= (x_{m-k-2s+1}^{(k)},\ldots,x_{m-k-s-1}^{(k)}) \quad \text{for } x \in \{a,b\}.
\end{aligned}
$$

If we have the QR factorization of $M_{s-1}^{(k)} = \left[\, A_{s-1}^{(k)} \;\middle|\; B_{s-1}^{(k)} \,\right]$, then the QR factorization of $M_s^{(k)}$ can then be computed in $O(s^2)$ flops by means of Givens rotations[9] as follows. Let $M_{s-1}^{(k)} = Q \cdot R$ with $R = \left[\, R(a) \;\middle|\; R(b) \,\right]$ partitioned as with $M_{s-1}^{(k)}$. Let $Q_1 = \operatorname{diag}[Q,1,1]$ and define $R_1 = Q_1^T M_s^{(k)}$. Then

$$
R_1 = \left[\begin{array}{c|c|c|c}
R_{11} & Q^T u(a) & R_{12} & Q^T u(b) \\
\hline
v(a) & a_{m-k-s+1}^{(k)} & v(b) & b_{m-k-s+1}^{(k)} \\
w(a) & a_{m-k-s}^{(k)} & w(b) & b_{m-k-s}^{(k)}
\end{array}\right].
$$

One then forms the product Q_2 of $s-2$ Givens rotations in order to zero out the "middle spike" in R_1, that is, the (i,s) entries of $R_2 = Q_2^T R_1$ are zero for $s+1 \le i \le 2s-2$. Then a product Q_3 of $2s-2$ Givens rotations yields $R_3 = Q_3^T R_2$ whose $(2s-1)$th row is zero everywhere but for its last two entries. A product Q_4 of $2s-1$ Givens rotations then leads to $R_4 = Q_4^T R_3$, which is upper triangular. The new QR decomposition is then given by $M_s^{(k)} = (Q_1 Q_2 Q_3 Q_4) R_4$. For example, when $s = 3$, the sparsity structure of the matrices R_1, R_2, R_3, R_4 is

$$
R_1 = \begin{bmatrix}
\times & \times & \cdot & \times & \times & \cdot \\
 & \times & \cdot & \times & \times & \cdot \\
 & & \cdot & \times & \times & \cdot \\
 & & \cdot & & \times & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix}
\quad \rightarrow \quad
R_2 = \begin{bmatrix}
\times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times \\
 & & & \times & \times & \times \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix}
$$

$$
\rightarrow \quad R_3 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \quad \rightarrow \quad R_4 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \\ & & & & & \times \end{bmatrix}.
$$

3.3 The case where $m \leq n$

The algorithm COPRIME has been designed for polynomial pairs whose first component is of greater degree than the second component. When $m < n$, it thus suffices to run the algorithm COPRIME with (b, a) in order to estimate $\epsilon(a, b) = \epsilon(b, a)$; when $m = n$, another lower bound can be obtained for $\epsilon(a, b)$ as follows.

Recall from [2] that

$$
\epsilon(a, b) = \inf_{z \in \overline{\mathbb{C}}} \Big|\Big| \Big(\frac{a(z)}{||(1, z^m)||}, \frac{b(z)}{||(1, z^n)||} \Big) \Big|\Big| \tag{12}
$$

where $\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$ and let this infimum be attained at some $z^* \in \overline{\mathbb{C}}$. Assume first that $|z^*| \leq 1$. It is shown in [2] that $\epsilon(a, b) = ||(a - a^*, b - b^*)||$ with $a^*(z) = a(z) - a(z^*)$ and $b^*(z) = b(z) - b(z^*)$. Now, let $\lambda = b_m/a_m$. One may assume w.l.o.g. that $|\lambda| \leq 1$, for otherwise it suffices to swap a and b. We then have $\deg(b - \lambda a) < \deg a = m$ and it follows from (12) that $\epsilon(a, b - \lambda a) \leq (1 + |\lambda|)\epsilon(a, b)$. In the case where $|z^*| > 1$, replacing in (12) z with z^{-1} and $a(z)$, $b(z)$ with respectively $\underline{a}(z)$, $\underline{b}(z)$, we further obtain $\epsilon(\underline{a}, \underline{b} - \lambda \underline{a}) \leq (1 + |\underline{\lambda}|)\epsilon(a, b)$. Here, $\underline{\lambda}$ is such that $|\underline{\lambda}| \leq 1$ and $\deg(\underline{b} - \underline{\lambda a}) < \deg \underline{a}$. (If $\deg \underline{a} > \deg \underline{b}$, we simply swap $\underline{a}$ and $\underline{b}$ and then take $\underline{\lambda} = 0$.) Hence the lower bound $1/\mu$ defined by

$$
\frac{1}{\mu} = \min \left\{ \frac{\epsilon(a, b - \lambda a)}{1 + |\lambda|}, \frac{\epsilon(\underline{a}, \underline{b} - \underline{\lambda}\, \underline{a})}{1 + |\underline{\lambda}|} \right\} \leq \epsilon(a, b). \tag{13}
$$

Since $\deg(b - \lambda a) < \deg a$, we can estimate $\epsilon(a, b - \lambda a)$ with the algorithm COPRIME. Another call to COPRIME yields an approximation of $\epsilon(\underline{a}, \underline{b} - \underline{\lambda})$.

When $1/\mu > 0$, the Bézout coefficients associated with the coprime pair (a, b) are $v - \lambda u$ and u with (v, u) being the Bézout coefficients associated with $(a, b - \underline{\lambda} a)$.

4 GCD-like operations

4.1 Approximate GCDs from the last accepted basis

Recall that a polynomial g is a **quasi-GCD[20] with precision** ϵ for a, b if there exist polynomials u_1, v_1, u_2, v_2 such that

$$
||(a, b) - g(u_2, v_2)|| < \epsilon, \quad ||av_1 + bu_1 - g|| < \epsilon ||g||, \quad \deg u_1 < m, \quad \deg v_1 < n.
$$

Also, g is an ϵ-**GCD**[7, 3] for a, b if there exist polynomials $\tilde{a}$, $\tilde{b}$ with degrees at most m, n such that $||(a, b) - (\tilde{a}, \tilde{b})|| \leq \epsilon$ and $g = \gcd(\tilde{a}, \tilde{b})$ has maximal degree.

All the computations in the COPRIME algorithm are done using finite precision arithmetic. As such there are residual error polynomials $\alpha^{(k)}$ and $\beta^{(k)}$ so that

$$
(a, b) = U^{(k)} \cdot (a^{(k)}, b^{(k)}) + (\alpha^{(k)}, \beta^{(k)})
$$

with $\deg a^{(k)} = m - k > \deg b^{(k)}$. Additionally, let $\rho_l(a,b)$ be the minimum of the set of all products $||(a,b)|| \cdot ||(g_a, g_b)^T||$ where the polynomial pair (g_a, g_b) is such that $\deg g_a < l$, $\deg g_b < l$ and

$$z^n a(z) g_a(z) + z^m b(z) g_b(z) = z^{m+n+l-1} + O(z^{m+n-1})_{z\to\infty}. \tag{14}$$

Beckermann and Labahn point out that when the last accepted basis has "small" errors then it can often still be used for computing either a quasi-GCD or an ϵ-GCD.

Theorem 4.1 *Let $U^{(k)}$ be the last well-behaved unimodular reduction computed by the algorithm COPRIME. Then*

(a) *If $||b^{(k)}|| + ||(\alpha^{(k)}, \beta^{(k)})|| < \epsilon |\det U^{(k)}(0)|/12$ with $0 < \epsilon < 1/6$ then $a^{(k)}$ is a quasi–GCD with precision ϵ.*

(b) *If $2||b^{(k)}|| + (2+4\rho_{n-m+2k}(a,b)) \cdot ||(\alpha^{(k)}, \beta^{(k)})|| \le \epsilon \, |\det U^{(k)}(0)|$ and $|\det U^{(k)}(0)| > 4\rho_{n-m+2k}(a,b) \cdot \epsilon$ then $a^{(k)}$ is an ϵ–GCD.*

□

4.2 Determining $\rho_l(a,b)$ via least squares

In order to check the conditions of Theorem 4.1, we need to estimate $\rho_l(a,b)$ (see Equation (14)). Computing a 2-norm equivalent of $\rho_l(a,b)$ allows to reduce to the following least squares problem.

When $l \ge 0$, consider the $l \times 2l$ complex matrix

$$M = \left[\begin{array}{ccc|ccc} a_m & & & b_n & & \\ \vdots & \ddots & & \vdots & \ddots & \\ a_{m-l+1} & \cdots & a_m & b_{n-l+1} & \cdots & b_n \end{array}\right].$$

In the case where $l < 0$, it suffices to replace l with $-l$ and $a(z)$, $b(z)$ with $z^m a(z^{-1})$, $z^n b(z^{-1})$ respectively. We then compute the Moore-Penrose inverse M^+ of M by singular value decomposition[9]. Then we may take for $\rho_l(a,b)$ the product $||M^+(.,1)||_2 \cdot ||(a,b)||_2$ where $M^+(.,1)$ denotes the first column of M^+.

5 Implementation

The techniques described so far have been implemented in a MAPLE package called SNAP (Symbolic-Numeric Algorithms for Polynomials). The list of the commands it offers is given below, where a and b denote univariate polynomials with floating-point coefficients.

- `AreCoprime`: coprimeness test for (a,b) when known up to a given ϵ;
- `DistanceToCommonDivisors`: estimate the distance between (a,b) and the set of polynomial pairs with at least one nontrivial divisor in common;
- `DistanceToSingularPolynomials`: estimate the distance between a and the set of polynomials with at least one mutiple root;
- `EpsilonGCD`: compute a polynomial g and a quantity ϵ such that g is an ϵ-GCD for (a,b);

- `EuclideanReduction`: return a degree reduced basis that is numerically equivalent to the basis (a, b);
- `IsSingular`: decide whether a has at least one mutiple root when known up to a given ϵ;
- `QuasiGCD`: compute a polynomial g and a quantity ϵ such that g is a quasi-GCD with precision ϵ for (a, b);
- `Quotient`: compute the quotient of a divided by b;
- `Remainder`: compute the remainder of a divided by b.

Except for `Quotient` and `Remainder`, for which we need only set up the set of linear equations induced by the operations and solve them numerically (see Section 2), the SNAP commands are related to Beckermann and Labahn's COPRIME algorithm of [3] as shown on Fig. 1.

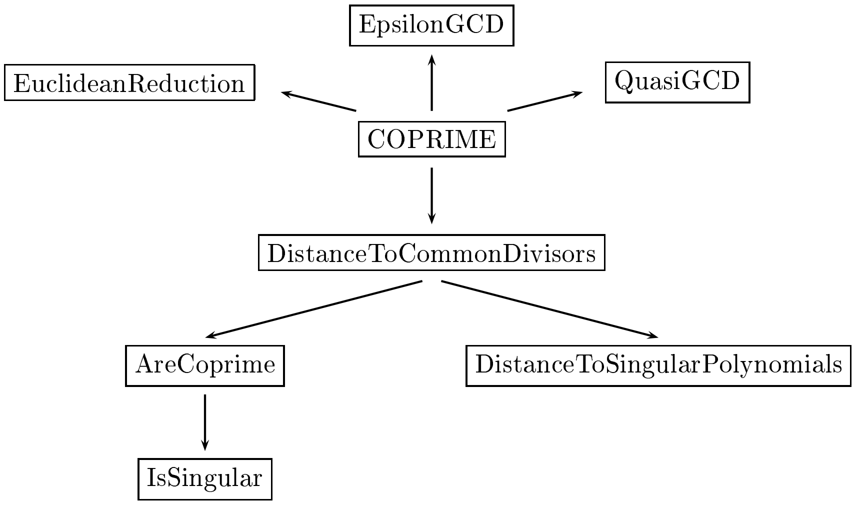

Figure 1: How SNAP relies on the algorithm COPRIME.

The command `DistanceToCommonDivisors` relies on COPRIME in order to provide the estimate $1/\kappa$ of (5) for the distance $\epsilon(a, b)$ of (4), whereas `EuclideanReduction` returns the last basis accepted by COPRIME. `EpsilonGCD` and `QuasiGCD` respectively implement parts (a) and (b) of Theorem 4.1; they are further used together with `DistanceToCommonDivisors` in the numeric coprimeness test `AreCoprime`. Note also that the commands `DistanceToSingularPolynomials` and `IsSingular` simply call the commands `DistanceToCommonDivisors` and `AreCoprime`, respectively, with b equal to the derivative of a. For each of these commands, several options are allowed; for example, the user can adjust the stability parameter τ of (8). We detail on such possibilities in the next paragraphs.

Concerning the arithmetic used, recall that MAPLE has two floating point systems, hardware float and software float. Software floats are used for extended precision arithmetic while hardware float makes use of double precision arithmetic available on all computers. Hardware floating point is much faster than software float, but is limited in its useage (see [16]). As such our implementation of COPRIME was done is such a way that it could run in either setting, making use of hardware floats where valid. In the case of hardware floating point arithmetic all the computations are done in double precision via a single call to `evalhf` and the output data is then converted back to user precision. When hardware float is not applicable (for example when the number of digits is set high) then the procedure is computed via the software float evaluator `evalf`. The structures allowed inside procedures that can work with hardware are limited (for example sets, lists and sequences are not allowed).

As a result of this `evalhf`-based approach, the SNAP package allows for numerical coprimeness to be detected efficiently for most inputs. Dense polynomials of degree of order 1000 can typically be handled by our implementation of COPRIME within a few minutes (on a Pentium III 800 MHz with 512MB of RAM, under Linux). This is illustrated in the table below. For comparison, we also give the time (in seconds) required to estimate $\epsilon(a, b)$ via setting up the Sylvester matrix $S(a, b)$ and solving the associated linear systems by LU decomposition (MAPLE's `LinearAlgebra[LinearSolve]` command).

$m = n + 1$	Sylvester	COPRIME
50	0.33	0.34
100	1.30	1.34
250	8.6	8.7
500	43	33
10^3	307	126
$2 \cdot 10^3$	3360	515

The remaining part of this section explains how to use the SNAP package and provides a description of each of the currently available functions.

5.1 Introduction to the SNAP package

Calling Sequence
`function`($args$)
`SNAP[function]`($args$)

Description
The SNAP (Symbolic-Numeric Algorithms for Polynomials) module provides tools for handling numeric polynomials in a stable and efficient way. The module consists of the following main components:

1. Approximate GCDs;
2. Coprimeness and singularity tests;
3. Distance problems;
4. Quotients and remainders.

Each function in the SNAP package can be accessed by using either the long form or the short form of the function name in the command calling sequence. For example, the coprimeness of two univariate numerical polynomials can be tested by using the following long-form calling sequence.
SNAP[AreCoprime](arguments)
The long form of the function name is necessary if the short form of the function has not been previously defined via with(SNAP, function), or if all of the package function short forms have not been previously defined via with(SNAP). Additionally, if another object in the current Maple session has the same name as a function in the SNAP package, the SNAP function can be accessed with the aid of unevaluation quotes, as SNAP['function'](arguments).

References
- B. Beckermann and G. Labahn, When are two numerical polynomials relatively prime? *Journal of Symbolic Computation 26* (1998) 677-689.
- B. Beckermann and G. Labahn, A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials, *Journal of Symbolic Computation 26* (1998) 691-714.

Examples
```
> with(SNAP):
```
1. Primality test
```
> a := 0.1*z^2+1.5*z-0.2;
```
$$a := 0.1z^2 + 1.5z - 0.2$$
```
> b := 0.2*z^3+0.15;
```
$$b := 0.2z^3 + 0.15$$
```
> AreCoprime(a,b,z,0.5);
```
$$false$$

```
> AreCoprime(a,b,z,0.1);
```

$$true$$

2. Approximate GCD computation (example from [19])

```
> a := (z^4-0.9999999)*(z^4-3.0000003*z-2.9999999);
```

$$a := (z^4 - 0.9999999)(z^4 - 3.0000003z - 2.9999999)$$

```
> b := (z^4-1.000001)*(z^3-3.000001*z+0.99999999);
```

$$b := (z^4 - 1.000001)(z^3 - 3.000001z + 0.99999999)$$

```
> EpsilonGCD(a,b,z);
```

$$0.4999999998z^4 + 0.2599345661 \cdot 10^{-7}z^3 + 0.2902317086 \cdot 10^{-6}z^2 + 0.4936931434 \cdot 10^{-6}z - 0.5000002917,$$

$$0.0006481143605$$

5.2 SNAP[AreCoprime]

Determine if two numeric polynomials are relatively prime up to a given error bound.

Calling Sequence

`AreCoprime`(a,b,z,ϵ,*out*)

Parameters

a	-	univariate numeric polynomial;
b	-	univariate numeric polynomial;
z	-	a name, the indeterminate for a and b;
ϵ	-	nonnegative numeric, error bound;
out	-	(optional) equation of the form output = obj where obj is 'BC' or a list containing one or more of this name; selects result objects to compute.

Description

The `AreCoprime`(a,b,z,ϵ) function checks whether univariate numeric polynomials a, b in z remain coprime after perturbations of order ϵ. Therefore we consider the set of polynomial pairs

$$\{(a^*, b^*) \,:\, \|(a - a^*, b - b^*)\| \le \epsilon,\, \deg a^* \le m,\, \deg b^* \le n\} \tag{15}$$

where $m = \deg a$ and $n = \deg b$. When $\epsilon = 0$, the pair (a, b) is considered to be exact and an exact GCD computation is done. Otherwise we compare $\epsilon > 0$ to the following two estimates of the distance $\epsilon(a, b)$ of (4): we recalled in Section 3 that $1/\kappa$ of (5) is a lower bound for $\epsilon(a, b)$; additionally it is not hard to verify that an upper bound μ is given by

$$\mu = \min(\|a\|, \|b\|) = \min(\sum_{i=0}^{m} |a_i|, \sum_{i=0}^{n} |b_i|).$$

Therefore, we return *true* when $\epsilon < 1/\kappa$, *false* when $\epsilon > \mu$ and *FAIL* otherwise; in the latter case, it is impossible to decide from estimates $1/\kappa$ and μ whether the set (15) has only coprime polynomial pairs or not.

The estimate $1/\kappa$ is obtained by calling the `SNAP[DistanceToCommonDivisors]` command (see Section 5.3). In the case where the algorithm COPRIME fails to produce a meaningful estimate but yields one of the approximate GCDs of Section 4.1, the output of `AreCoprime` is *false*; if no approximate GCD can be guaranteed, we return *FAIL*.

The output option (out) determines the content of the returned expression sequence. Depending on what is included in the output option, every occurrence of BC yields the list $[v, u]$ of the Bézout coefficients for a and b, i.e. the numeric polynomials in z that satisfy (2). This list is empty when the answer is either *false* or *FAIL*.

Examples

```
> with(SNAP):
> a := 0.1*z^2+1.5*z-0.2;
```

$$a := 0.1z^2 + 1.5z - 0.2$$

```
> b := 0.2*z^3+0.15;
```

$$b := 0.2z^3 + 0.15$$

```
> AreCoprime(a,b,z,0.5);
```

$$false$$

```
> AreCoprime(a,b,z,0.1);
```

$$true$$

```
> AreCoprime(a,b,z,0.1,output='BC');
```

$$true, [-0.871004100817765581z^2 - 0.112232907263963163z - 0.05851459268, 0.435502050408882734z + 6.588647210]$$

```
> expand(a * %[2][1] + b * %[2][2]);
```

$$1.000000000 - 0.1 \cdot 10^{-9} z^2$$

See also SNAP[DistanceToCommonDivisors], SNAP[DistanceToSingularPolynomials]

5.3 SNAP[DistanceToCommonDivisors]

Lower bound on the distance between a pair of univariate polynomials and the set of univariate polynomial pairs with a common root.

Calling Sequence

`DistanceToCommonDivisors`(a, b, z, $tau = \tau$, out)

Parameters

a	-	univariate numeric polynomial;
b	-	univariate numeric polynomial;
z	-	a name, the indeterminate for a and b;
$tau = \tau$	-	(optional) equation where τ is of type numeric and nonnegative; stability parameter;
out	-	(optional) equation of the form output = obj where obj is one of 'LB', 'BC', 'E', 'LAB', 'QGCD', 'RB' or 'UR', or a list containing one or more of these names; selects result objects to compute.

Description

The `DistanceToCommonDivisors`(a,b,z) function computes the estimate $1/\kappa$ associated with the distance $\epsilon(a, b)$ to the set of numeric polynomial pairs with at least one common root. (See (5)).

When the input polynomials have "small" degrees (typically $m + n \leq 600$), we simply solve the linear systems (11) with the MAPLE command `LinearSolve`. Otherwise, the method used is the fast and weakly stable algorithm COPRIME recalled in Section 3.1; When no reliable estimate $1/\kappa$ can be obtained, the cause of failure can be displayed by setting `infolevel[DistanceToCommonDivisors]` to 5.

The optional stability parameter tau (see (8)) can be set to any nonnegative value τ in order to monitor the quality of the output: decreasing τ yields a more reliable solution; increasing τ reduces the risk of failure.

The output option (out) determines the content of the returned expression sequence. Depending on what is included in the output option, an expression sequence containing one or more of the following quantities can be returned:

- The lower bound LB is a nonnegative float or FAIL.
- The list BC is equal to the Bézout coefficients $[v, u]$ associated with (a, b), i.e. the numeric polynomials in z that satisfy (2). The list [ ] is returned instead if and only if the algorithm COPRIME failed to compute a reliable lower bound.
- The list LAB contains the last basis of (6), say $(a^{(k_0)}, b^{(k_0)})$, accepted by COPRIME.
- E can take the following values: the residual error for the computed lower bound (which allows to verify the numerical quality of the solution); if no reliable lower bound was computed and if an ϵ-GCD g for (a, b) was found, then E $= \epsilon$ and $g = a^{(k_0)}$ (i.e. the first component of the last accepted basis) when $\deg a > \deg b$, and $g = b^{(k_0)}$ (i.e. the second component of the last accepted basis) when $\deg a < \deg b$; if no ϵ-GCD for (a, b) could be found or if $\deg a = \deg b$ then E $= FAIL$.

- QGCD can take the following values: the precision of the quasi-GCD found; in this case, the quasi-GCD is $a^{(k_0)}$, the first component of the last accepted basis. When no quasi-GCD can be found, QGCD is assigned to $FAIL$.
- RB is the list of the indices of all the bases $(a^{(k)}, b^{(k)})$ of (6) which were rejected by COPRIME.
- UR contains a 2 by 2 unimodular matrix polynomial $U(z)$ such that $(a, b) \cdot U = (a^{(k_0)}, b^{(k_0)})$.

Note that choosing any of the above options but BC implies that, even for small degree polynomials, the method used is the algorithm COPRIME.

Examples

```
> with(SNAP):
> a := z^2+3.1*z-2;
```

$$a := z^2 + 3.1z - 2$$

```
> b := 2*z^3+1.5;
```

$$b := 2z + 1.5$$

```
> DistanceToCommonDivisors(a,b,z);
```

$$0.8761833704$$

```
> DistanceToCommonDivisors(a,b,z,tau=1e-2,output=['LB','BC','LAB','E','RB']);
```

$$0.8761833704, [-0.1446350696 - 0.1246262639z - 0.2654882420z^2, 0.4738199072 + 0.1327441210z],$$
$$[0., 0.4999999999], 0.265799999\dot{6}10^{-7}, [\,]$$

```
> expand(a*%[2][1]+b*%[2][2]);
```

$$-0.37 \cdot 10^{-8} z^2 + 0.3 \cdot 10^{-9} z^3 + 1.000000000 - 0.65 \cdot 10^{-8} z$$

See also SNAP[AreCoprime], SNAP[DistanceToSingularPolynomials], SNAP[EpsilonGCD], SNAP[QuasiGCD]

5.4 SNAP[DistanceToSingularPolynomials]

Lower bound on the distance between a univariate polynomial and the set of univariate polynomials with a double root.

Calling Sequence

`DistanceToSingularPolynomials`(a, b, z, $tau = \tau$, out)

Parameters

a	-	univariate numeric polynomial
b	-	univariate numeric polynomial
z	-	a name, the indeterminate for a and b
$tau = \tau$	-	(optional) equation where τ is of type numeric and nonnegative; stability parameter
out	-	(optional) equation of the form output = obj where obj is one of 'LB', 'BC', 'E', 'LAB', 'QGCD', 'RB' or 'UR', or a list containing one or more of these names; selects result objects to compute

Description

For a given numeric polynomial a in z, the `DistanceToSingularPolynomials`(a,z) function returns an estimate of the distance between a and the set of univariate singular polynomials, i.e. the set of complex polynomials with a double root. This distance is clearly equal to $\epsilon(a, da/dz)$.

The method used is to call `SNAP[DistanceToSingularPolynomials]`(a,b,z) with $b(z) = da(z)/dz$.

All the optional arguments of `SNAP[DistanceToSingularPolynomials]` are also available for this command.

Examples

```
> with(SNAP):
> a := z^2+3.1*z-2;
```

$$a := z^2 + 3.1z - 2$$

```
> DistanceToSingularPolynomials(a,z);
```

$$0.5985723997$$

```
> b := diff(a,z);
```

$$b := 2z + 3.1$$

```
> DistanceToCommonDivisors(a,b,z);
```

$$0.5985723997$$

See also SNAP[DistanceToCommonDivisors]

5.5 SNAP[EpsilonGCD]

Epsilon-GCD for a pair of univariate numeric polynomials.

Calling Sequence

`EpsilonGCD`(a, b, z, $tau = \tau$)

Parameters

a	-	univariate numeric polynomial
b	-	univariate numeric polynomial
z	-	a name, the indeterminate for a and b
$tau = \tau$	-	(optional) equation where τ is of type numeric and nonnegative; stability parameter

Description

The `EpsilonGCD`(a,b,z) function returns a univariate numeric polynomial g together with a positive float ϵ such that g is an ϵ-GCD for the input polynomials (a, b) (see Section 4.1). If no such approximate GCD can be certified by means of Theorem 4.1 then *FAIL* is returned.

The optional stability parameter *tau* (see (8)) can be set to any nonnegative value τ in order to monitor the quality of the output: decreasing τ yields a more reliable solution; increasing τ reduces the risk of failure.

Examples

```
> with(SNAP):
> a := (z^4-0.9999999)*(z^4-3.0000003*z-2.9999999);
```

$$a := (z^4 - 0.9999999)(z^4 - 3.0000003z - 2.9999999)$$

```
> b := (z^4-1.000001)*(z^3-3.000001*z+0.99999999);
```

$$b := (z^4 - 1.000001)(z^3 - 3.000001z + 0.99999999)$$

```
> EpsilonGCD(a,b,z);
```

$$0.4999999998z^4 + 0.2599345661 \cdot 10^{-7}z^3 + 0.2902317086 \cdot 10^{-6}z^2 + 0.4936931434 \cdot 10^{-6}z - 0.5000002917,$$

$$0.0006481143605$$

See also SNAP[DistanceToCommonDivisors], SNAP[QuasiGCD]

5.6 SNAP[EuclideanReduction]

The smallest degree pair of univariate polynomials obtained by Euclidean-like unimodular reduction.

Calling Sequence
`EuclideanReduction`(a, b, z, *tau* $= \tau$, *out*)

Parameters

a	-	univariate numeric polynomial
b	-	univariate numeric polynomial
z	-	a name, the indeterminate for a and b
$tau = \tau$	-	(optional) equation where τ is of type numeric and nonnegative; stability parameter
out	-	(optional) equation of the form output = obj where obj is 'UR' or a list containing one or more of this name; selects result object to compute

Description

The `EuclideanReduction`(a,b,z) function returns the last basis of (6), say $(a^{(k_0)}, b^{(k_0)})$, accepted by the algorithm COPRIME.

The optional stability parameter *tau* can be set to any nonnegative value τ in order to monitor the quality of the output: decreasing τ yields a more reliable solution; increasing τ reduces the degrees of the returned basis.

Depending on what is included in the output option, an expression sequence containing one or more of the following quantity can be returned: UR contains a 2 by 2 unimodular matrix polynomial $U(z)$ such that $(a, b) \cdot U = (a^{(k_0)}, b^{(k_0)})$.

Examples

```
> with(SNAP):
> a := z^6-12.4*z^5+62.53*z^4-163.542*z^3+232.9776*z^2-170.69184*z+50.18112;
```

$$a := z^6 - 12.4z^5 + 62.53z^4 - 163.542z^3 + 232.9776z^2 - 170.69184z + 50.18112$$

```
> b := z^5-17.6*z^4+118.26*z^3-372.992*z^2-274.09272+538.3333*z;
```

$$b := z^5 - 17.6z^4 + 118.26z^3 - 372.992z^2 + 538.3333z - 274.09272$$

```
> EuclideanReduction(a,b,z);
```

$$[4.000000000z^4 - 45.32014526z^3 + 182.6434982z^2 - 301.3056473z + 164.9022927,$$

$$3.489992909z^3 - 25.44123904z^2 + 55.50550385z - 34.91733545]$$

```
> EuclideanReduction(a,b,z,tau=1e-8);
```

$$[0.2500000000z^2 - 0.8750003765z + 0.6600005057, -0.907264 \cdot 10^{-7}z + 0.1335296 \cdot 10^{-6}]$$

See also SNAP[DistanceToCommonDivisors], SNAP[EpsilonGCD]

5.7 SNAP[IsSingular]

Determine if a numeric polynomial has a double root up to a given error bound.

Calling Sequence

`IsSingular`(a,z,ϵ,out)

Parameters

a	-	univariate numeric polynomial
z	-	a name, the indeterminate for a
ϵ	-	nonnegative numeric, error bound
out	-	(optional) equation of the form output = obj where obj is 'BC' or a list containing one or more of this name; selects result objects to compute

Description

The `IsSingular`(a,z,ϵ) function checks whether the polynomial $a(z)$ has no double root up to perturbation ϵ by calling `AreCoprime`(a,b,z,ϵ) with $b = da/dz$. Hence the answer can be *true*, *false* or *FAIL*.

The output option (out) determines the content of the returned expression sequence. Depending on what is included in the output option, every occurrence of BC yields the list $[v, u]$ of the Bézout coefficients for a and da/dz, i.e. the numeric polynomials in z that satisfy (2) with $b = da/dz$. This list is empty when the answer is either *false* or *FAIL*.

Examples

```
> with(SNAP):
> a := 0.00103*z^3 - 0.00195*z^2 - 0.00501 * z + 0.00604;
```

$$a := 0.00103z^3 - 0.00195z^2 - 0.00501z + 0.00604$$

```
> IsSingular(a,z,1e-1);
```

$$true$$

```
> IsSingular(a,z,1e-2);
```

$$FAIL$$

```
> IsSingular(a,z,1e-4);
```

$$false$$

See also SNAP[AreCoprime]

5.8 SNAP[QuasiGCD]

Schönhage's quasi-GCD for a pair of univariate numeric polynomials

Calling Sequence

`QuasiGCD`(a, b, z, $tau = \tau$)

Parameters

a	-	univariate numeric polynomial
b	-	univariate numeric polynomial
z	-	a name, the indeterminate for a and b
$tau = \tau$	-	(optional) equation where τ is of type numeric and nonnegative; stability parameter

Description

The `QuasiGCD`(a,b,z) function returns a univariate numeric polynomial g together with a positive float ϵ such that g is an quasi-GCD of precision ϵ for the input polynomials (a, b) (see Section 4.1). If no such approximate GCD can be certified by means of Theorem 4.1 then *FAIL* is returned.

The optional stability parameter *tau* (see (8)) can be set to any nonnegative value τ in order to monitor the quality of the output: decreasing τ yields a more reliable solution; increasing τ reduces the risk of failure.

References

A. Schönhage, Quasi–GCD computations, *J. Complexity* **1**(1985) 118-137.

Examples

```
> with(SNAP):
> a := (z^4-0.9999999)*(z^4-3.0000003*z-2.9999999);
```

$$a := (z^4 - 0.9999999)(z^4 - 3.0000003z - 2.9999999)$$

```
> b := (z^4-1.000001)*(z^3-3.000001*z+0.99999999);
```

$$b := (z^4 - 1.000001)(z^3 - 3.000001z + 0.99999999)$$

```
> QuasiGCD(a,b,z);
```

$$0.4999999998z^4 + 0.2599345661 \cdot 10^{-7}z^3 + 0.2902317086 \cdot 10^{-6}z^2 + 0.4936931434 \cdot 10^{-6}z - 0.5000002917,$$

$$0.0005263828643$$

See also SNAP[DistanceToCommonDivisors], SNAP[EpsilonGCD]

5.9 SNAP[Quotient], SNAP[Remainder]

Quotient and remainder of two numeric polynomials.

Calling Sequence

`Quotient`(a, b, z)
`Quotient`(a, b, z, 'r')
`Remainder`(a, b, z)
`Remainder`(a, b, z, 'q')

Parameters

a	-	univariate numeric polynomial
b	-	univariate numeric polynomial
z	-	a name, the indeterminate for a and b
q, r	-	(optional) unevaluated names

Description

The `Quotient` function returns the quotient of $a(z)$ divided by $b(z)$. The `Remainder` function returns the remainder of $a(z)$ divided by $b(z)$. The quotient $q(z)$ and remainder $r(z)$ satisfy $\deg r < \deg b$ and $\|a - (b \cdot q + r)\|$ is small.

If a fourth argument is included in the calling sequence for `Remainder` or `Quotient`, it will be assigned the quotient $q(z)$ or remainder $r(z)$, respectively.

The method used is the one of Section 2. We solve the associated linear system via QR decomposition with the MAPLE function `LinearAlgebra[LinearSolve]`.

Examples

```
> with(SNAP):
> a := -85*z^5-55*z^4-37*z^3-35*z^2+97*z+50;
```

$$a := -85z^5 - 55z^4 - 37z^3 - 35z^2 + 97z + 50$$

```
> b := 79*z^3+56*z^2+49*z+63;
```

$$b := 79z^3 + 56z^2 + 49z + 63$$

```
> q := Quotient(a,b,z,'r');
```

$$-1.07594936708860733z^2 + 0.0664957538855951714z + 0.1518703389$$

```
> r;
```

$$21.02177920z^2 + 85.36912090z + 40.43216865$$

```
> expand(a-b*q-r);
```

$$-0.1 \cdot 10^{-7}z^5 - 0.1 \cdot 10^{-7}z^4 - 0.1 \cdot 10^{-7}z^3 - 0.1 \cdot 10^{-8}$$

References

[1] B. Beckermann, The stable computation of formal orthogonal polynomials, *Numerical Algorithms* **11** (1996) 1-23.

[2] B. Beckermann and G. Labahn, When are two numerical polynomials relatively prime? *Journal of Symbolic Computation* **26** (1998) 677-689.

[3] B. Beckermann and G. Labahn, A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials, *Journal of Symbolic Computation* **26** (1998) 691-714.

[4] J.R. Bunch, The weak and strong stability of algorithms in numerical linear algebra, *Lin. Alg. Appl.* **88-89** (1987) 49-66.

[5] S. Chandrasekaran and A.H. Sayed, A fast stable solver for nonsymmetric Toeplitz and quasi-Toeplitz systems of linear equations, *SIAM J. Matrix Anal. Appl.* **19(1)** (1998) 107-139.

[6] R.M. Corless, P.M. Gianni, B.M. Trager and S.M. Watt, The singular value decomposition for polynomial systems, Proceedings ISSAC'95, ACM Press (1995) 195-207.

[7] I. Z. Emiris, Symbolic-numeric algebra for polynomials. Survey available at `http://www-sop.inria.fr/galaad/emiris` (1997).

[8] I. Z. Emiris, A. Galligo and H. Lombardi, Certified approximate univariate GCDs, *J. Pure and Applied Algebra* **117** (1997) 229-251.

[9] G.H. Golub and C. Van Loan, Matrix computations, Johns Hopkins University Press (1989).

[10] N.J. Higham, *Accuracy and Stability of Numerical Algorithms* (SIAM, Philadelphia, 1996).

[11] V. Hribernig and H.J. Stetter, Detection and validation of clusters of polynomial zeros, J. Symbol. Comput. 24(6) (1997) 667-681.

[12] C.P. Jeannerod and G. Labahn, SNAP User's Guide (2002).

[13] N. Karmarkar and Y.N. Laksmann, Approximate polynomial greatest common divisors and nearest singular polynomials, Proceedings ISSAC'96, ACM Press (1996) 35-43.

[14] N. Karmarkar and Y.N. Laksmann, On approximate GCDs of univariate polynomials, *Journal of Symbolic Computation* **26** (1998) 653-666.

[15] T. Kailath, Linear systems, Prentice–Hall (1980).

[16] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter, J. McCarron and P. DeMarco, *Maple 7 Programming Guide,* Toronto: Waterloo Maple Inc., 2001.

[17] M.-T. Noda & T. Sasaki, Approximate GCD and its applications to ill–conditioned algebraic equations, *J. Comput. Appl. Math.* **38** (1991) 335-351.

[18] V.Y. Pan, Numerical computations of a polynomial GCD and extensions. Technical Report 2969, INRIA, Sophia-Antipolis (1996).

[19] D. Rupprecht, Eléments de géométrie algébrique approchée: étude du PGCD et de la factorisation, Thèse de l'Université de Nice-Sophia Antipolis (2000).

[20] A. Schönhage, Quasi–GCD computations, *J. Complexity* **1** (1985) 118-137.

[21] H. Stetter, Numerical Polynomial Algebra: Concepts and Algorithms, ATCM 2000, Proceed. 5th Asian Technology Conf. in Math. (Eds.: W.-Ch. Yang, S.-Ch. Chu, J.-Ch. Chuan), 22-36, ATCM Inc. USA (2000).

[22] C.J. Zarowski, X. Ma and F.W. Fairman, QR-factorization method for computing the greatest common divisor of polynomials with inexact coefficients, *IEEE Trans. Signal Processing* **48(11)** (2000) 3042-3051.