Algebraiska kurvor

Peter Waher

Innehåll

Kapitel 1

Symboler

De symboler som använts i detta dokument har kategoriserats och listats här nedan.

1.1 Mängder

$\varnothing$	Den tomma mängden
$\in$	... är ett element i ...
$\notin$	... är inte ett element i ...
$\cup$	Union av två mängder
$\cap$	Skärning av två mängder
$\setminus$	Komplement av den högra mänden i den vänstra
$\subset$	... är en strikt delmängd av ..., dvs de är inte lika
$\subseteq$	... är en delmängd av ..., och de kan vara lika
$\{\ldots\}$	Beskrivning av en mängd
$\|\ldots\|$	Antalet element i mängden
$\langle\ldots\rangle$	Mängden genererad av ...

1.2 Fördefinierade mängder

$\mathbb{N}$	De naturliga talen: $1, 2, 3, \ldots$
$\mathbb{Z}^+$	De positiva heltalen inklusive noll: $0, 1, 2, 3, \ldots$
$\mathbb{Z}$	Alla heltal
$\mathbb{R}$	De reella talen
$\mathbb{C}$	De komplexa talen

1.3 Heltal

$\mathbb{Z}_m$	$\{[n]_m : n \in \mathbb{Z}\}$ (skrivs även $\mathbb{Z}/\langle m \rangle$ eller $\mathbb{Z}/m$)
$[n]_m$	$\{a \in \mathbb{Z} : a \equiv n \pmod{m}\}$
$a \equiv n \pmod{m}$	$\exists b \in \mathbb{Z} : a = n + b \cdot m$
$a \mid b$	a delar b, dvs $\exists n \in \mathbb{N} : b = a \cdot n$
$a \nmid b$	a delar inte b

1.4 Logiska operatorer

$\exists$	Existens av ...
$\forall$	För alla ...
$\Longrightarrow$	Logisk implikation
$\Longleftrightarrow$	Logisk ekvivalens
$\vee$	Logiskt eller
$\wedge$	Logiskt och
$\neg$	Logisk negation
$\therefore$	Därför, logisk slutsats

1.5 Övrigt

0	Nollelementet i gruppen som behandlas. Kan även stå som förkortning av $\{0\}$.
1	Ettelementet i ringen som behandlas. Kan även stå som förkortning av $\{1\}$.
$\mathbf{0}$	$(0, \ldots, 0) \in \mathbb{C}^n$
∞	Oändligheten. Vilken oändlighet framgår av sammanhanget.

Kapitel 2

Kurvor

2.1 Introduktion

Definition 2.1 *En* ***plan kurva*** *C är en delmängd i $\mathbb{C}^2$ sådan att det finns två kontinuerliga funktioner $f : \mathbb{R} \mapsto \mathbb{C}$ och $g : \mathbb{R} \mapsto \mathbb{C}$ sådana att $C = \{(f(t), g(t)) : t \in \mathbb{R}\}$. (f, g) är en* ***parametrisering*** *av C. Om C kan parametriseras av två analytiska funktioner f och g kallas C* ***analytisk****. Om den kan parametriseras av två polynom kallas C för* ***algebraisk****. Om den kan parametriseras av två formella potensserier kallas C* ***algebroid****.*

Eftersom vi kommer att studera kurvor lokalt, kommer de olika termerna nämnda ovan inte att skilja åt så mycket: Polynom är ju i sig formella potensserier där bara ett ändligt antal koefficienter är nollskiljda. Analytiska funktioner kan också skrivas som formella potensserier som konvergerar i ett område kring den punkt vi vill studera.

För att förenkla notationen kan vi identifiera kurvan C med en viss parametrisering (f, g), även om parametriseringen inte är unik. Detta görs enklast genom att identifiera kurvan med funktionen $C : \mathbb{R} \mapsto \mathbb{C}^2, C(t) = (f(t), g(t))$. Notera dock att kurvan som sådan och en av dess parametriseringar är två olika objekt.

När vi studerar egenskaper hos plana kurvor kan vi anta att kurvan går genom origo, samt att $f(0) = g(0) = 0$, eller enklare skrivet $C(0) = \mathbf{0}$. Om så inte är fallet kan vi först göra en omparametrisering av C och sedan ett enkelt byte av koordinatsystem:

Anta att (f, g) är en parametrisering av C och att vi vill studera kurvan kring punkten (x_0, y_0), och att $t = t_0$ är sådan att $x_0 = f(t_0)$ och $y_0 = g(t_0)$. Omparametriseringen kan vi göra på följande sätt:

$$\begin{cases} f^*(t) & = & f(t + t_0) \\ g^*(t) & = & g(t + t_0) \\ C^*(t) & = & (f^*(t), g^*(t)) \end{cases}$$

Detta ger att $C^*(\mathbb{R}) = C(\mathbb{R})$, och kurvorna C^* och C är identiska. Koordinatbytet gör man enklast:

$$\begin{cases} x^{**} & = & x - x_0 \\ y^{**} & = & y - y_0 \\ f^{**}(t) & = & f^*(t) - x_0 \\ g^{**}(t) & = & g^*(t) - y_0 \\ C^{**}(t) & = & (f^{**}(t), g^{**}(t)) \end{cases}$$

Detta ger $C^{**}(\mathbb{R}) = C^*(\mathbb{R}) - (x_0, y_0) = C(\mathbb{R}) - (x_0, y_0)$ och $C^{**}(0) = \mathbf{0}$. I fortsättningen antar vi därför att de plana kurvor C som vi studerar har egenskapen $C(0) = \mathbf{0}$.

Definition 2.2 *Om en kurva C har en parametrisering (f, g) sådan att $f'(0) \neq 0$ eller $g'(0) \neq 0$ kallas kurvan* ***regulär****. Annars kallas kurvan* ***singulär****.*

Bara för att $f'(0) = 0$ och $g'(0) = 0$ i en parametrisering (f, g) av en kurva C, betyder inte detta att kurvan är singulär. Det kan ju finnas en parametrisering av samma kurva, där någon av derivatorna är nollskilda. Exempelvis är (t^3, t^3) och (t, t) är två olika parametriseringar av samma kurva. I det första exemplet är derivatorna 0 i origo, medan i det andra exemplet är de båda nollskilda. (Notera: (t^2, t^2) och (t, t) motsvarar inte samma kurva, eftersom den första inte innehåller punkterna $(t, t), t < 0$.)

2.2 Omparametrisering av kurvor

För att studera egenskaper hos en viss kurva $C = (f(t), g(t))$ kan det ibland vara intressant att göra en omparametrisering av kurvan på en enklare eller mer fördelaktig form. För utritande av kurvor spelar parametriseringen inte så stor roll. Men vill man beräkna $y(x) = g(f^{-1}(x))$ eller $x(y) = f(g^{-1}(y))$ för motsvarande kurva, står man genast inför en mängd problem eftersom parametriseringen kan vara så generell. Ofta är det väldigt svårt att explicit hitta inversen av en analytisk funktion, även om den är så “enkel” som ett polynom. Dessutom brukar inverserna vara flervärda så man måste på något sätt bestämma sig för vilken del av kurvan som man är intresserad av.

Vi kan väldigt lätt skapa omparametriseringar av kurvan C ovan, genom att ta en funktion $\phi(t)$ som är analytisk kring $t = 0$ och sådan att $\phi(0) = 0$, och därefter skapa parametriseringen

$$C^*(t) = (f^*(t), g^*(t)) = (f(\phi(t)), g(\phi(t)))$$

Om $\phi(t), f(t)$ och $g(t)$ är realvärda är också parametriseringen $C^*(t)$ realvärd.

Eftersom ϕ är analytisk kring $t = 0$ och $\phi(0) = 0$ vet vi att den är ett-till-ett i ett område kring $t = 0$. Detta säger oss att $\exists I_1 \subseteq \mathbb{R}, I_2 \subseteq \mathbb{R}$ sådana att $C^*(I_1) = C(I_2)$, dvs. kurvan som den nya parametriseringen $(f^*(t), g^*(t))$ bildar och den gamla $(f(t), g(t))$ överenstämmer "delvis". Är dessutom ϕ ett-till-ett i hela $\mathbb{R}$ gäller att $C^*(\mathbb{R}) = C(\mathbb{R})$ eller att kurvorna överenstämmer "helt".

Eftersom alla funktioner ϕ som är analytiska kring $t = 0$ och $\phi(0) = 0$ kan skrivas på formen

$$\phi(t) = \sum_{k=1}^{\infty} a_k t^k$$

i ett område kring $t = 0$ (även $f(t)$ och $g(t)$ har denna form) skall vi studera våra möjligheter att skapa omparametriseringar av en kurva $C(t)$ m.h.a. potensserier.

Först några definitioner:

Definition 2.3 ***Ordningen*** *av ett polynom eller en potensserie* $f(t) = \sum a_i t^i$ *är det minsta heltalet* k *sådant att koefficienten* a_k *är nollskiljd, och skrivs* $\mathbf{o}(f)$. ***Graden*** *for motsvarande polynom är det största heltalet* k *sådant att koefficienten* a_k *inte är noll.*

Om $\phi(t) = \sum a_k t^k$ och $f(t) = \sum b_k t^k$, hur ser då $f^*(t) = f(\phi(t)) = \sum c_k t^k$ ut? Följande vet vi: $\mathbf{o}(f^*) = \mathbf{o}(f(\phi)) = \mathbf{o}(f) \cdot \mathbf{o}(\phi)$, samt att

$$c_{\mathbf{o}(f^*)} = b_{\mathbf{o}(f)} \cdot {a_{\mathbf{o}(\phi)}}^{\mathbf{o}(f)}$$

Om vi vill att $f^*(t)$ skall vara på formen $f^*(t) = t^n$, där $n = \mathbf{o}(f) \cdot \mathbf{o}(\phi)$, så måste således

$$a_{\mathbf{o}(\phi)} = \left(\frac{1}{b_{\mathbf{o}(f)}}\right)^{\frac{1}{\mathbf{o}(f)}} \neq 0$$

Betraktar man hela $\mathbb{C}$ finns $\mathbf{o}(f)$ olika lösningar för $a_{\mathbf{o}(\phi)}$. Dock skall vi försöka välja en lösning sådan att alla a_k blir reella (förutsatt att alla b_k är reella).

För att göra beräkningarna lite enklare och överskådligare, kan vi tills vidare begränsa oss att bara betrakta funktioner $\phi(t)$ sådana att $a_1 \neq 0$, dvs $\mathbf{o}(\phi)$=1. Detta medför att $n = \mathbf{o}(f^*) = \mathbf{o}(f)$.

Den andra koefficienten c_{n+1} i f^* får följande utseende:

$$c_{n+1} = b_{n+1} {a_1}^{n+1} + n b_n {a_1}^{n-1} a_2$$

Eftersom $n \neq 0$, $b_n \neq 0$ och $a_1 \neq 0$ är detta en linjär ekvation i a_2 med unik lösning. Eftersom vi vill att f^* skall vara på formen $f^*(t) = t^n$ sätter vi således $c_{n+1} = 0$, och vi får:

$$a_2 = -\frac{b_{n+1}{a_1}^{n+1}}{nb_n{a_1}^{n-1}} = -\frac{b_{n+1}}{nb_n}{a_1}^2$$

Vi antar nu att $a_1 \dots a_m$ är beräknade och att $c_{n+1} = \dots = c_{n+m-1} = 0$. För att beräkna a_{m+1} tittar vi på c_{n+m}, och sätter denna till 0. c_{n+m} har bidrag från de $m+1$ lägsta potenserna i $f(\phi(t))$: $b_n\phi(t)^n$, $b_{n+1}\phi(t)^{n+1}$, $\dots$, $b_{n+m}\phi(t)^{n+m}$.

Från $b_{n+m}\phi(t)^{n+m}$ fås ett bidrag $b_{n+m}{a_1}^{n+m}$, som är en konstant (vi har ju antagit att $a_1 \dots a_m$ är givna). Alla andra termer i denna potens har en grad $> m+n$.

Från $b_{n+m-k}\phi(t)^{n+m-k}$ skapas en mängd termer på formen

$$b_{n+m-k} \cdot \prod_{j=1}^{n+m-k} a_{i_j} t^{i_j} = b_{n+m-k} \cdot \left(\prod_{j=1}^{n+m-k} a_{i_j} \right) t^{\sum i_j}$$

för olika talserier $\{i_j\}_1^{n+m-k}$ där $i_j \in \mathbb{N}$ (dvs $i_j \geq 1$). Till koefficienten c_{n+m} bidrar bara termer sådana att $\sum i_j = n+m$. Eftersom $i_j \geq 1$ får vi att $\sum i_j \geq n+m-k$, oavsett val av $\{i_j\}$. $\sum i_j = n+m-k$ fås om $i_j = 1, \forall j$. Det högsta a_i som kan förekomma i en term som ovan för c_{n+m} är således a_{k+1}. Sådana termer fås om man väljer alla i_j utom en till 1 och den resterande till $k+1$. Därför har dessa termer formen $b_{n+m-k}{a_1}^{n+m-k-1}a_{k+1}t^{n+m}$. Nämnda val av i_j kan endast göras på $n+m-k$ olika sätt.

Av ovanstående resonemang ser man att bidragen till c_{n+m} från termerna $b_{n+m-k}\phi(t)^{n+m-k}$ är konstanta (bara beroende av $a_1, \dots, a_m$), för $k = 0, \dots, m-1$, och att bidraget från $b_n\phi(t)^n$ (dvs. $k = m$) till c_{n+m} är på formen

$$A_m(a_1, \dots, a_m) + b_n n {a_1}^{n-1} a_{m+1}$$

där $A_m(a_1, \dots, a_m)$ är en konstant (beroende på de givna $a_1, \dots, a_m$. Om vi kallar bidragen från $b_{n+m-k}\phi(t)^{n+m-k}$ till c_{n+m} för $B_m(a_1, \dots, a_m)$ får vi alltså:

$$c_{n+m} = A_m(a_1, \dots, a_m) + B_m(a_1, \dots, a_m) + nb_n{a_1}^{n-1}a_{m+1}$$

Eftersom $b_n \neq 0$, $n \neq 0$ och $a_1 \neq 0$ ser vi att ekvationen $c_{n+m} = 0$ har en unik lösning:

$$a_{m+1} = -\frac{A_m(a_1, \dots, a_m) + B_m(a_1, \dots, a_m)}{nb_n{a_1}^{n-1}}$$

Värt att observera är följande: $A_m(a_1, \dots, a_m)$ och $B_m(a_1, \dots, a_m)$ beror bara på koefficienterna $a_1, \dots, a_m$ och b_k. Om alla dessa koefficienter är reella blir således också a_{m+1} reell.

Vi är nu redo för följande sats.

Sats 2.1 *Om $C = C(t) = (f(t), g(t))$ är en analytisk, algebroid eller algebraisk kurva och $f(t)$ och $g(t)$ är realvärda, samt $f(0) = g(0) = 0$, kan kurvan C omparametriseras på formen $C^*(t) = (\pm t^n, g^*(t))$ eller på formen $C^*(t) = (f^*(t), \pm t^n)$ i ett intervall kring $t = 0$, där $f(t)$ och $g(t)$ är formella potensserier. Dessutom gäller att $\mathbf{o}(f^*) \geq n$ eller $\mathbf{o}(g^*) \geq n$.*

Bevis:

Vi kan utan att begränsa oss anta att $f(t)$ och $g(t)$ kan skrivas som potensserier kring $t = 0$. Vi antar först att $n = \mathbf{o}(f) \leq \mathbf{o}(g)$.

Från resonemanget ovan ser vi att vi kan skapa en formell potensserie $\phi(t) = \sum a_k t^k$ sådan att $f^*(t) = f(\phi(t)) = b_n {a_1}^n t^n = c \cdot t^n$. Vi vill välja a_1 sådan att $c = \pm 1$.

Om n är udda, eller $b_n > 0$ låter vi $c = 1$ vilket ger oss

$$a_1 = \left(\frac{1}{b_n}\right)^{\frac{1}{n}}$$

Vi väljer här huvudgrenen i funktionen $x^{1/n}$, vilket ger oss ett reellt tal a_1. Om nu n är jämn och $b_n < 0$, låter vi $c = -1$ vilket ger oss

$$a_1 = \left(\frac{1}{-b_n}\right)^{\frac{1}{n}}$$

Även detta tal är ett reellt tal. Om $f(t)$ är realvärd består dess potensserie bara av realvärda koefficienter ($\{b_k\}$). Eftersom a_1 är realvärd fås då med induktion att alla a_k också är realvärda. Detta ger oss att $g^*(t) = g(\phi(t))$ också är realvärd. Dessutom fås att

$$\mathbf{o}(g^*) = \mathbf{o}(g) \cdot \mathbf{o}(\phi) = \mathbf{o}(g) \cdot 1 = \mathbf{o}(g) \geq \mathbf{o}(f)$$

Om nu $\mathbf{o}(f) > \mathbf{o}(g)$, väljer vi $\phi(t)$ sådan att $g^*(t) = g(\phi(t)) = \pm t^n$. Resten av beviset är identiskt med ovanstående, vilket avslutar beviset för första delen av satsen.

$\mathcal{VSB}$

I Bilaga B finns beskrivet en algoritm som beräknar ovanstående parametrisering för en godtycklig algebraisk kurva, upp till en maximal grad.

Kapitel 3

Semigrupper

Vid studier av singulariteter hos algebraiska kurvor använder man ofta motsvarande semigrupp som invariant för klassificering. Innan vi fördjupar oss i just detta måste vi först nämna några talteoretiska definitioner och satser, och därefter fördjupa oss lite i semigrupper.

3.1 Heltal modulo p

Definition 3.1 *Heltalen $m \in \mathbb{N}$ och $n \in \mathbb{N}$ sägs vara **relativt prima** om $m \geq 2$, $n \geq 2$ samt $p \mid m \ \wedge \ p \mid n \Longrightarrow p = 1$*

Att m och n är relativt prima är identiskt med att säga att $\operatorname{sgd}(m, n) = 1$ (största gemensamma delaren).

Lemma 3.1 *Om m och n är relativt prima och $0 < a < m$ gäller att $[a \cdot n]_m \neq 0$*

BEVIS:

Antag att $[a \cdot n]_m = 0 \ \wedge \ a > 0$:

$$\begin{aligned} [a \cdot n]_m = 0 \quad &\Longrightarrow \quad \exists b \in \mathbb{N} \ : \ a \cdot n = b \cdot m \\ &\Longrightarrow \quad m \mid a (\text{eftersom } \operatorname{sgd}(m, n) = 1) \\ &\Longrightarrow \quad a \geq m \end{aligned}$$

$\mathcal{VSB}$

Lemma 3.2 *Om m och n är relativt prima och $0 < a, b < m$ gäller:*

$$[an]_m = [bn]_m \Longleftrightarrow a = b$$

BEVIS:

Vi kan utan att begränsa oss antaga att $a \geq b$.

$$\begin{aligned} [an]_m = [bn]_m &\iff [(a-b)\cdot n]_m = 0 \\ &\iff a-b=0 \text{ (eftersom } 0 \leq a-b < m \text{ samt lemma 3.1)} \end{aligned}$$

$\mathcal{VSB}$

3.2 Semigrupper

Definition 3.2 *För en* ***semigrupp*** *G gäller:*

$$\begin{aligned} 0 &\in G \\ a \in G \wedge b \in G &\Longrightarrow (a+b) \in G \end{aligned}$$

Notera skillnaden mellan en *semigrupp* och en *grupp*: Alla element i en grupp har även en additativ invers, dvs. i gruppen tillåts subtraktion. I en semigrupp är det tillräckligt med bara addition av element.

Definition 3.3 *En serie tal $n_1, \ldots, n_k$* ***genererar*** *semigruppen*[1] *G om*

$$G = \left\{ \sum_{i=1}^{k} a_i \cdot n_i \ : \ a_i \in \mathbb{Z}^+ \right\}$$

Detta skrivs även $G = \langle n_1, \ldots, n_k \rangle$.

Sats 3.1 *Om m och n är relativt prima innehåller semigruppen $G = \langle m, n \rangle$ alla tal större än eller lika med $c = (m-1)(n-1)$, men inte $c-1$.*

BEVIS:

Först betraktar vi hur $[n]_m$ genererar hela $\mathbb{Z}_m$. Från lemma 3.1 vet vi:

$$\begin{aligned} [n]_m &\neq 0 \\ [2n]_m &\neq 0 \\ \vdots\ &\ \vdots\ \ \vdots \\ [(m-1)\cdot n]_m &\neq 0 \end{aligned}$$

[1]Notera att det är viktigt att särskilja mellan generation av semigrupper och grupper. Vid generering av grupper gäller att $a_i \in \mathbb{Z}$. Det finns ingen annan skillnad i notationen.

Lemma 3.2 säger dessutom att dessa $m-1$ värdena är unika. Vi har alltså $m-1$ st unika nollskilda element i $\mathbb{Z}_m$, i vilket det bara finns $m-1$ nollskilda element. Alltså:

$$\mathbb{Z}_m = \bigcup_{i=0}^{m-1} \{i \cdot [n]_m\}$$

I följande resonemang kan vi utan att begränsa oss anta att $m < n$. För att beräkna det högsta värdet som inte är med i $\langle m, n\rangle$ använder vi oss av uteslutningsmetoden:

Först delar vi upp $\mathbb{Z}$ i delar om m element vardera:

$$0 \ldots m-1, \quad m \ldots 2m-1, \quad \ldots$$

Sedan stryker vi helt enkelt tal efter tal tills inte fler finns att stryka bort.

Vi börjar med att stryka bort talet n, och alla motsvarigheter i $\mathbb{Z}_m$ $(n+m, n+2m, \ldots)$. Därefter fortsätter vi med $2n$ och dess motsvarigheter i $\mathbb{Z}_m$ $(2n+m, 2n+2m, \ldots)$, etc.

Eftersom vi vet att $[n]_m$ helt genererar $\mathbb{Z}_m$ vet vi att det sista tal som vi stryker är $(m-1)n$ och dess motsvariheter i $\mathbb{Z}_m$: $(m-1)n+m, (m-1)n+2m, \ldots$. Därefter är alla element som är med i $\langle m, n\rangle$ strukna ur $\mathbb{Z}$.

Den del av $\mathbb{Z}$ som innehåller det sista talet vi strök innehåller inte några andra ostrukna tal, eftersom $m < n$. Det högsta ostrukna talet, och därför också det högsta tal som inte är element i $\langle m, n\rangle$ är således $(m-1)n - m$, dvs motsvarigheten till $(m-1)n$ i den del som föregår delen där $(m-1)n$ finns. Vi kan nu beräkna c:

$$c = (m-1)n - m + 1 = mn - n - m + 1 = (m-1)(n-1)$$

$\mathcal{VSB}$

3.3 Numeriska semigrupper

Definition 3.4 *En* ***numerisk semigrupp*** G *är en speciell form av semigrupp, där även följande villkor gäller:*

$$G \subseteq \mathbb{Z}^+$$
$$\|\mathbb{Z}^+ \backslash G\| < \infty$$

Lemma 3.3 $G = \langle m, n\rangle$ *i sats (3.1) är en numerisk semigrupp.*

Bevis:

$$\begin{aligned}
0 &= 0 \cdot m + 0 \cdot n \in G \\
a \in G \wedge b \in G &\iff \exists c, d, e, f \in \mathbb{Z}^+ : a = cm + dn \wedge b = em + fn \\
&\implies a + b = (c+e)m + (d+f)n \in G \\
G &\subset \mathbb{Z}^+ \text{ (per definition av } \langle m, n \rangle) \\
\|\mathbb{Z}^+ \backslash G\| &\leq c - 1 < \infty
\end{aligned}$$

$\mathcal{VSB}$

Definition 3.5 *I varje numerisk semigrupp G finns det ett tal c_G sådant att följande villkor uppfylls:*

$$\begin{aligned}
c_G &\in G \\
n > c_G &\implies n \in G \\
c_G - 1 &\notin G
\end{aligned}$$

c_G kallas för ***konduktören*** *för G. c i sats (3.1) är konduktör för den numeriska semigruppen $G = \langle m, n \rangle$.*

Sats 3.2 *Om semigruppen $G = \langle n_1, \ldots, n_k \rangle$ är numerisk så är den största gemensamma delaren av talen, $sgd(n_1, \ldots, n_k) = 1$.*

Bevis:

Vi antar motsatsen: $d = \mathrm{sgd}(n_1, \ldots, n_k) \geq 2$.

$$\begin{aligned}
n \in G &\implies \exists a_i \in \mathbb{Z}^+ : n = \sum_{i=1}^{k} a_i n_i \\
d \mid n_i, \forall i &\implies d \mid n \\
&\implies \mathrm{sgd}(G) = d \\
d \geq 2 &\implies d \nmid a \cdot d + 1 \\
&\implies \{a \cdot d + 1 : a \in \mathbb{Z}^+\} \cap G = \varnothing \\
&\implies \{a \cdot d + 1 : a \in \mathbb{Z}^+\} \subseteq \mathbb{Z}^+ \backslash G \\
&\implies \|\mathbb{Z}^+ \backslash G\| = \infty
\end{aligned}$$

$\therefore \mathrm{sgd}(n_1, \ldots, n_k) = 1$

$\mathcal{VSB}$

Sats 3.3 *Varje numerisk semigrupp G har ett minimalt generatorsystem $n_1, \ldots, n_k$, dvs. $G = \langle n_1, \ldots, n_k \rangle$. Detta system är unikt för G.*

BEVIS:

Först skall vi bevisa att det finns en ändlig mängd tal i G som genererar G.

Eftersom G är numerisk betyder detta att det finns ett största tal $N \in \mathbb{Z}^+ : (n > N \Longrightarrow n \in G)$. Välj a och b så att $2^a > N \wedge 3^b > N$. Detta ger att $2^a \in G \wedge 3^b \in G$. Men $\text{sgd}(2^a, 3^b) = 1$ vilket enligt sats 3.1 ger att $\left\langle 2^a, 3^b \right\rangle$ innehåller alla tal större än eller lika med $c = (2^a - 1)(3^b - 1)$.

$\therefore \{1, \ldots, c-1\} \cap G$ genererar G.

Då vi vet att G har ändliga generatorer kan vi skapa mängden $\widehat{M}$ som mängden av alla ändliga generatorer, och $\widehat{M}' = \{\|M\| : M \in \widehat{M}\}$. Låt $k = \inf \widehat{M}'$. Eftersom $\widehat{M}'$ bara innehåller positiva heltal existerar k samt $\exists M_- \in \widehat{M} : \|M_-\| = k$.

Antag nu att vi har ett annat generatorsystem N_- sådant att $\|N_-\| = k$. Låt $0 < n_1 < \ldots < n_k$ och $0 < m_1 < \ldots < m_k$ vara tal sådana att $N_- = \{n_1, \ldots, n_k\}$ och $M_- = \{m_1, \ldots, m_k\}$.

Uppenbarligen är n_1 och m_1 lika med det minsta nollskilda talet i G. Därför är $n_1 = m_1$.

Antag nu att $n_1 = m_1, \ldots, n_i = m_i$. Eftersom N_- är ett minimalt generatorsystem så är $n_{i+1} \notin \langle n_1, \ldots, n_i \rangle$. På samma sätt gäller att $m_{i+1} \notin \langle m_1, \ldots, m_i \rangle$. Låt $m \in G$ vara det minsta talet i $G \setminus \langle n_1, \ldots, n_i \rangle$. Eftersom m är det minsta sådana talet gäller att $n_{i+1} \geq m$. Men n_{i+1} kan inte vara större än m heller eftersom detta skulle innebära att alla tal $n_{i+2}, \ldots, n_k$ också skulle vara större än m och m kan inte vara summan av ett större tal och ett element ur $\langle n_1, \ldots, n_i \rangle$.

$\therefore n_{i+1} = m$.

På samma sätt härleder man att $m_{i+1} = m = n_{i+1}$. Med induktion fås att $n_i = m_i, \forall i$.

$\mathcal{VSB}$

Sats 3.3 säger inte bara att det finns ett unikt minimalt generatorsystem $\langle n_1, \ldots, n_k \rangle$ för den numeriska semigruppen G, utan ger också en metod för att beräkna den (även om denna metod kan vara väldigt krävande):

1. Låt n_1 vara det minsta nollskilda elementet i G.

2. Om $n_1, \ldots, n_i$ är beräknade, låt n_{i+1} vara det minsta talet i $G \setminus \langle n_1, \ldots, n_i \rangle$.

3. Fortsätt med punkt 2 tills hela G är genererad.

Sats 3.2 säger dessutom att för $n_1, \ldots, n_k$ skapade på detta sätt gäller $\text{sgd}(n_1, \ldots, n_k) = 1$.

3.4 Beräkning av konduktören från generatorerna

Sats 3.4 *Om $n_1, \ldots, n_k$ är heltal sådana att $sgd(n_1, \ldots, n_k) = 1$ så gäller att $G = \langle n_1, \ldots, n_k \rangle$ är en numerisk semigrupp.*

BEVIS:

Det enda vi behöver bevisa är att G har en konduktör. Vi kan i detta bevis anta, utan att begränsa oss att $n_1 < \ldots < n_k$.

Precis som i sats (3.1) delar vi upp $\mathbb{Z}^+$ i segment om n_1 element vardera:

$$0 \ldots n_1 - 1, \quad n_1 \ldots 2n_1 - 1, \quad \ldots$$

$$\begin{aligned} sgd(n_1, \ldots, n_k) = 1 \quad &\Longrightarrow \quad \exists A_i \in \mathbb{Z} : \sum_{i=1}^{k} A_i n_i = 1 \implies \\ &\Longrightarrow \quad \sum_{i=1}^{k} [A_i]_{n_1} n_i = [1]_{n_1} \implies \\ &\Longrightarrow \quad \left\langle [n_1]_{n_1}, \ldots, [n_k]_{n_1} \right\rangle = \mathbb{Z}_{n_1} \end{aligned}$$

Precis som i sats (3.1) börjar vi sedan med uteslutningsmetoden att stryka de tal i $\mathbb{N}$ som genereras av $\{n_i\}$. Först stryker vi alla multiplar av n_1. Därefter alla linjära kombinationer av n_1 och n_2, osv tills vi strukit alla linjära kombinationer av alla n_i.

Eftersom hela $\mathbb{Z}_{n_1}$ genereras helt av ett ändligt antal generatorer vet vi att efter ett ändligt antal segment är alla tal i dessa resterande segment helt strukna, dvs de ingår i $\langle n_1, \ldots, n_k \rangle$.

$\therefore$ Endast ett ändligt antal tal ingår inte i G.

$\therefore$ G har en konduktör.

$\mathcal{VSB}$

Vi har antagit i detta bevis att det är känt att $\exists A_i : \sum A_i n_i = sgd(\{n_i\})$. I Bilaga B finns några procedurer som visar hur man kan beräkna dessa $\{A_i\}$ givet $\{n_i\}$.

Vi kan lätt göra ett litet tillägg till sats (3.4) för att skapa oss en algoritm för att beräkna denna konduktör. Algoritmen går till som följer:

1. Först skapar vi en lista $\mathbb{Z}_{\min}$ med n_1 heltal alla satta till 0. Denna motsvarar $\mathbb{Z}_{n_1}$. Vi antar fortfarande att n_1 är den minsta av generatorerna (annars fungerar inte algoritmen).

2. En slinga går igenom generatorerna $n_1, \ldots, n_k$.

3. För varje generator n_i skapar vi en ny slinga som genererar multiplarna $x_{ij} = jn_i$, tills dess att alla multiplarna $[jn_i]_{n_1}$ har genererats. Detta behöver göras $n_i/\text{sgd}(n_1, n_i)$ gånger.

4. För varje element a i listan $\mathbb{Z}_{\min}$ tittar vi i listan på position $a + x_{ij}$ mod n_1. Om där står 0 eller ett annat tal som är större än $a + x_{ij}$ fyller vi den med värdet $a + x_{ij}$, annars låter vi det vara.

5. Vi tittar även i listan på position $x_{ij} \mod n_1$. Om där står 0 eller ett annat tal som är större än x_{ij} fyller vi den med värdet x_{ij}, annars låter vi det vara.

6. När slingorna 2 till 5 har genomförts är hela $\mathbb{Z}_{\min}$ fylld av positiva heltal, enligt sats (3.4). Vi går då igenom $\mathbb{Z}_{\min}$ och ser vilket tal där som är störst. Vi kallar detta tal för m.

7. Konduktören c blir således $c = m - n_1 + 1$ (enligt samma resonemang som i sats (3.1)).

För att se exempel på denna algoritm, och även en algoritm för beräkning av semigrupper från dess generatorer, se Bilaga B.

Bilaga A

Reparametrize - källtexter till Maple

Reparametrize-funktionen omparametriserar en algebraisk kurva givet på formen $C(t) = (f(t), g(t))$ till formen $(\pm t^n, g^*(t))$, där $n <= \mathbf{o}(g^*)$, eller till formen $(f^*(t), \pm t^n)$, där $n <= \mathbf{o}(f^*)$. Funktionen använder två hjälpfunktioner; *Composition* och *MakePoly*, vilka beskrivs först. Sist i detta appendix följer några exempel på hur *Reparametrize* kan användas i Maple.

A.1 MakePoly

MakePoly tar en lista (array) av koefficienter och returnerar ett polynom i den valda variabeln. Ingående parametrar är:

$\mathit{MakePoly}(a,\ \mathit{Variable})$

a	En lista (array) med koefficienter.
Variable	Namnet på variabeln som skall användas i polynomet.

$$
\begin{array}{l}
\mathit{MakePoly} := \textbf{proc}(a,\ \mathit{Variable}) \\
\quad \textbf{local}\, i,\ r; \\
\quad\quad r := 0\,; \\
\quad\quad \textbf{for}\ i\ \textbf{to}\ \mathrm{nops}(a)\ \textbf{do if}\ \mathrm{op}(i,\ a) \neq 0\ \textbf{then} \\
\quad\quad\quad r := r + \mathrm{op}(i,\ a) \times \mathit{Variable}^{(i-1)}\ \textbf{fi} \\
\quad\quad \textbf{od}\,; \\
\quad\quad \mathrm{RETURN}(\mathrm{sort}(r)) \\
\quad \textbf{end}
\end{array}
$$

A.2 Composition

Composition beräknar kompositionen $f(g(t))$ av två polynom $f(t)$ och $g(t)$. Kompositionen beräknas bara upp till en viss grad för att spara på minne och tid.

Anledningen till att vi bara vill beräkna kompositionen $f(g(t))$ till en viss grad är att vi kanske redan vet att $f(t)$ och $g(t)$ bara har en viss ordning av noggrannhet. I detta fall kommer kompositionen att ha samma noggrannhet, och inte noggrannheten i kvadrat (vilken kan vara avsevärd).

För att göra det enklare att komma åt koefficienterna i den resulterande kompositionen, returneras resultatet som en array i stället för ett polynom.

De ingående parametrarna är:

Composition(f, g, *Variable*, *MaxOrder*)

f	Polynom f
g	Polynom g
MaxOrder	Beräkning sker endast till denna ordning.

$Composition$:= **proc**(f, g, $Variable$, $MaxOrder$)
  local gc, gcn, $gcnt$, i, j, k, r, c, $floop$;
    $gc :=$ array(0..$MaxOrder$) ;
    $gcn :=$ array(0..$MaxOrder$) ;
    $gcnt :=$ array(0..$MaxOrder$) ;
    $r :=$ array(0..$MaxOrder$) ;
    for i **from** 0 **to** $MaxOrder$ **do**
      $gc_i :=$ coeff(g, $Variable$, i) ; $gcn_i := gc_i$; $r_i := 0$
    od;
    $floop :=$ degree(f) ;
    if $MaxOrder < floop$ **then** $floop := MaxOrder$ **fi** ;
    $r_0 :=$ coeff(f, $Variable$, 0) ;
    for i **to** $floop$ **do**
      $c :=$ coeff(f, $Variable$, i) ;
      for j **from** 0 **to** $MaxOrder$ **do** $r_j := r_j + c \times gcn_j$ **od** ;
      if $i < MaxOrder$ **then**
        for j **from** 0 **to** $MaxOrder$ **do**
          $gcnt_j := 0$; **for** k **from** 0 **to** j **do**
            $gcnt_j := gcnt_j + gc_k \times gcn_{j-k}$
          od
        od;
        for j **from** 0 **to** $MaxOrder$ **do** $gcn_j := gcnt_j$ **od**
      fi

```
    od;
    RETURN(eval(r))
end
```

A.3 Reparametrize

Reparametrize tar en parametrisering av en kurva och skapar en ny parametrisering enligt algoritmen i texten. Parametrarna som krävs är följande:

Reparametrize(x, y, *Variable*, *t0*, *MaxOrder*, *Branch*, *AllowNegation*)

x	Parametrisering av x-komponenten av kurvan.
y	Parametrisering av y-komponenten av kurvan.
Variable	Namnet på variabeln i parametriseringen (motsvarande t i texten ovan.
t0	Runt vilken punkt kurvan skall parametriseras.
MaxOrder	Till vilken ordning omparametriseringen skall göras.
Branch	Vilken gren som skall väljas som lösning. Standard är 0.
AllowNegation	Om omparametriseringen får innehålla $-t^n$ eller inte. Standard är *true*.

```
Reparametrize := proc(x, y, Variable, t0, MaxOrder, Branch, AllowNegation)
    local xt, yt, a, p, q, x0, y0, i, j, k, shifted, temp, sols, StartTime, MinOrder,
        arg, z, Negate;

    StartTime := time() ;
    if not type(Branch, integer) then
        ERROR(`Branch must be an integer!`, Branch)
    fi;
```

if $t0 \neq 0$ **then**
$xt :=$ collect(expand(convert(taylor(
collect(expand(subs($Variable = Variable + t0,\ x$)), $Variable$),
$Variable = 0,\ MaxOrder + 1$), $polynom$)), $Variable$);
$yt :=$ collect(expand(convert(taylor(
collect(expand(subs($Variable = Variable + t0,\ y$)), $Variable$),
$Variable = 0,\ MaxOrder + 1$), $polynom$)), $Variable$)
else
$xt :=$ collect(expand(convert(taylor($x,\ Variable = 0,\ MaxOrder + 1$),
$polynom$)), $Variable$);
$yt :=$ collect(expand(convert(taylor($y,\ Variable = 0,\ MaxOrder + 1$),
$polynom$)), $Variable$);
fi;
$x0 :=$ coeff($xt,\ Variable,\ 0$);
$y0 :=$ coeff($yt,\ Variable,\ 0$);
$xt := xt - x0$;
$yt := yt - y0$;
$MinOrder :=$ ldegree($xt,\ Variable$);
$shifted :=$ ldegree($yt,\ Variable$) $< MinOrder$;
if $shifted$ **then**
$temp := xt$; $xt := yt$; $yt := temp$; $MinOrder :=$ ldegree($xt,\ Variable$)
fi;
$a :=$ array($1..MaxOrder - MinOrder + 1$);
$z := 1/$coeff($xt,\ Variable,\ MinOrder$);
if $z < 0$ **and** $AllowNegation$ **then** $xt := -xt$; $z := -z$; $Negate := true$
else $Negate := false$
fi;
$arg :=$ argument(z);
$a_1 :=$ abs$(z)^{(1/MinOrder)} \times ($
$\cos((2 \times (MinOrder - 2 + Branch) \times \pi + arg)/MinOrder)$
$+ I \times \sin((2 \times (MinOrder - 2 + Branch) \times \pi + arg)/MinOrder))$;
$p := a_1 \times Variable$;
for j **from** 2 **to** $MaxOrder - MinOrder + 1$ **do** $p := p + a_j \times Variable^j$ **od**;
$q :=$ Composition($yt,\ p,\ Variable,\ MaxOrder$);
$p :=$ Composition($xt,\ p,\ Variable,\ MaxOrder$);
$i := MinOrder + 1$;
$j := 2$;

while $i \le \mathit{MaxOrder}$ **do**
$\mathit{sols} := [\mathrm{solve}(p_i = 0,\, a_j)]\,;$
$q_i := \mathrm{expand}(\mathrm{subs}(a_j = \mathit{sols}_1,\, q_i))\,;$
if $i < \mathit{MaxOrder}$ **thenfor** k **from** $i+1$ **to** $\mathit{MaxOrder}$ **do**
$p_k := \mathrm{subs}(a_j = \mathit{sols}_1,\, p_k)\,;\; q_k := \mathrm{subs}(a_j = \mathit{sols}_1,\, q_k)$
od
fi;
$a_j := \mathit{sols}_1\,;$
$i := i + 1\,;$
$j := j + 1$
od;
$\mathit{xt} := q_0\,;$
for i **to** $\mathit{MaxOrder}$ **do if** $q_i \ne 0$ **then** $\mathit{xt} := \mathit{xt} + q_i \times \mathit{Variable}^i$ **fi od**;
if $\mathit{shifted}$ **then**
$\mathit{xt} := \mathit{x0} + \mathit{xt}\,;$
if Negate **then** $\mathit{yt} := \mathit{y0} - \mathit{Variable}^{\mathit{MinOrder}}$
else $\mathit{yt} := \mathit{y0} + \mathit{Variable}^{\mathit{MinOrder}}$
fi
else
$\mathit{yt} := \mathit{y0} + \mathit{xt}\,;$
if Negate **then** $\mathit{xt} := \mathit{x0} - \mathit{Variable}^{\mathit{MinOrder}}$
else $\mathit{xt} := \mathit{x0} + \mathit{Variable}^{\mathit{MinOrder}}$
fi
fi;
printf("Elapsed Time: %0.3f s.\n", time() − $\mathit{StartTime}$);
RETURN([xt, yt])
end

A.4 Exempel 1

Kurvan (t^2+t^3, t^5+t^6) har en singularitet i $t = 0$. Dessutom passerar kurvan genom origo då $t = -1$. Först ber vi Maple att parametrisera om kurvan kring $t = 0$:

```
> Reparametrize(t^2+t^3,t^5+t^6,t,0,10,0,false);
```

```
Elapsed Time: .060 s.
```

$$[t^2,\, t^5 - \frac{3}{2}\, t^6 + \frac{21}{8}\, t^7 - 5\, t^8 + \frac{1287}{128}\, t^9 - 21\, t^{10}]$$

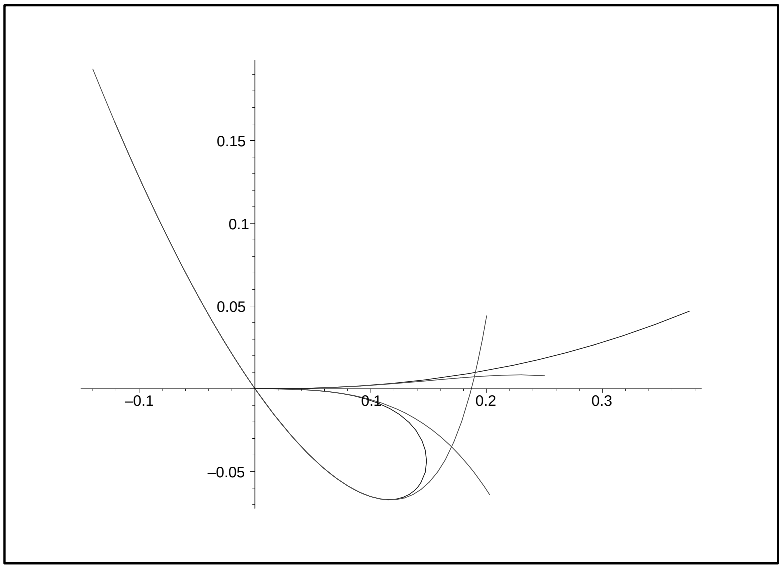
Därefter vill vi ha en omparametrisering kring $t = -1$:

```
> Reparametrize(t^2+t^3,t^5+t^6,t,-1,10,0,false);
```

```
Elapsed Time: .065 s.
```

$$[t,\ -t+3\,t^2+3\,t^3+10\,t^4+42\,t^5+198\,t^6+1001\,t^7+5304\,t^8+29070\,t^9+163438\,t^{10}]$$

Ritar vi ut orginalparametriseringen av kurvan och de två omparametriseringarna i samma graf får vi:



A.5 Exempel 2

Ellipsen $(3\sin(t),\, 2\cos(t))$ saknar singulariteter:

```
> Reparametrize(3*sin(t),2*cos(t),t,0,10,0,false);
```

```
Elapsed Time: .075 s.
```

$$[t,\ 2-\frac{1}{9}\,t^2-\frac{1}{324}\,t^4-\frac{1}{5832}\,t^6-\frac{5}{419904}\,t^8-\frac{7}{7558272}\,t^{10}]$$

```
> Reparametrize(3*sin(t),2*cos(t),t,Pi,10,0,false);
```

```
Elapsed Time: .105 s.
```

$$[t,\ -2+\frac{1}{9}\,t^2+\frac{1}{324}\,t^4+\frac{1}{5832}\,t^6+\frac{5}{419904}\,t^8+\frac{7}{7558272}\,t^{10}]$$

```
> Reparametrize(3*sin(t),2*cos(t),t,Pi/2,10,0,false);
```

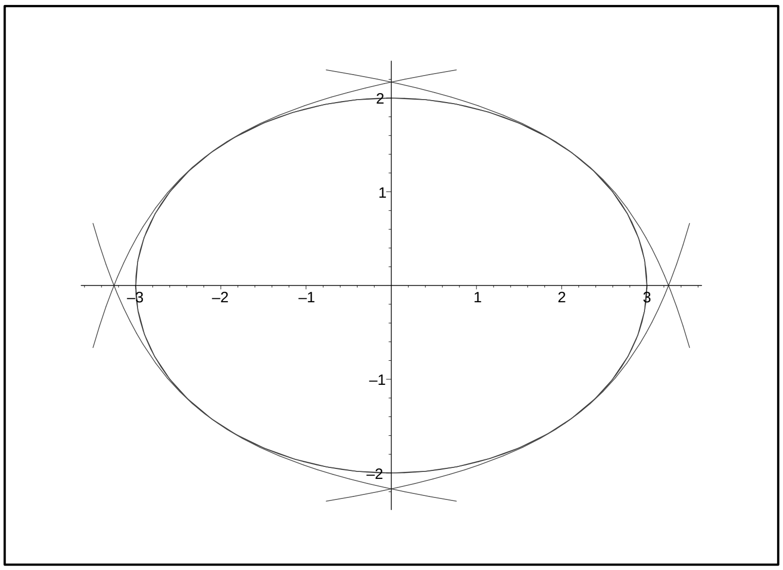
```
Elapsed Time: .070 s.
```

$$[3 - \frac{3}{8}\,t^2 - \frac{3}{128}\,t^4 - \frac{3}{1024}\,t^6 - \frac{15}{32768}\,t^8 - \frac{21}{262144}\,t^{10},\, t]$$

```
> Reparametrize(3*sin(t),2*cos(t),t,-Pi/2,10,0,false);
```

```
Elapsed Time: .070 s.
```

$$[-3 + \frac{3}{8}\,t^2 + \frac{3}{128}\,t^4 + \frac{3}{1024}\,t^6 + \frac{15}{32768}\,t^8 + \frac{21}{262144}\,t^{10},\, t]$$



A.6 Exempel 3

Kurvan $\left(t^3\,(t-1)^3\,(t+1)^3, t^5\,(t-1)^2\,(t+1)^2\right) = (t^9 - 3\,t^7 + 3\,t^5 - t^3,\, t^5 - 2\,t^7 + t^9)$ har en singuläritet i $t = 0$ av ordning 2, en i $t = 1$ av ordning 1 och en i $t = -1$ av ordning 1.

```
> Reparametrize(t^3*(t-1)^3*(t+1)^3,t^5*(t-1)^2*(t+1)^2,
     t,0,15,1,false);
```

```
Elapsed Time: .230 s.
```

$$[t^3,\, -t^5 - 3\,t^7 - 12\,t^9 - 55\,t^{11} - 273\,t^{13} - 1428\,t^{15}]$$

```
> Reparametrize(t^3*(t-1)^3*(t+1)^3,t^5*(t-1)^2*(t+1)^2,
     t,1,15,0,false);
```

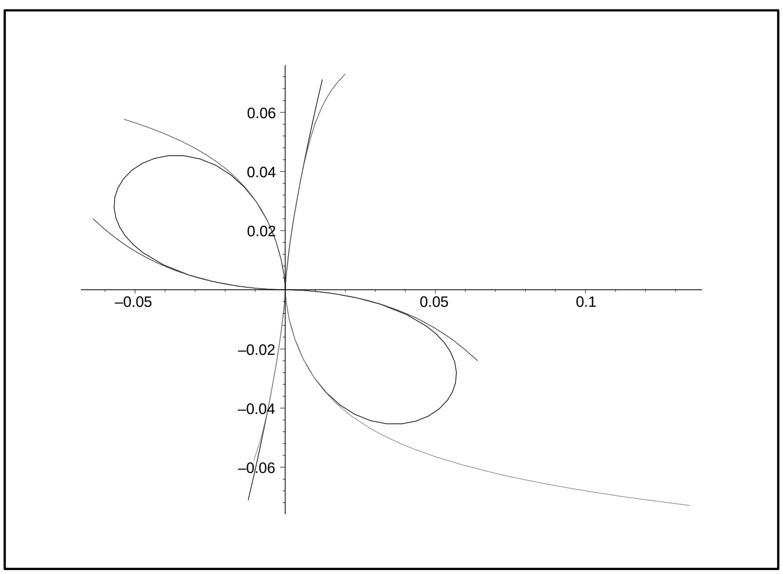
```
Elapsed Time: 2.805 s.
```

$$[\frac{1}{2}\sqrt{4}\,t^3 - \frac{9}{4}\,t^4 + \frac{207}{64}\,\sqrt{4}\,t^5 - 21\,t^6 + \frac{150183}{4096}\,\sqrt{4}\,t^7 - \frac{137655}{512}\,t^8 + \frac{66893079}{131072}\,\sqrt{4}\,t^9 - 3978\,t^{10} + \frac{132735945771}{16777216}\,\sqrt{4}\,t^{11} - \frac{8385901667}{131072}\,t^{12} + \frac{70379121262905}{536870912}\,\sqrt{4}\,t^{13} - \frac{4345965}{4}\,t^{14} + \frac{78087826643607459}{34359738368}\,\sqrt{4}\,t^{15},\ t^2]$$

```
> Reparametrize(t^3*(t-1)^3*(t+1)^3,t^5*(t-1)^2*(t+1)^2,
    t,-1,15,0,true);
```

```
Elapsed Time: 2.695 s.
```

$$[\frac{1}{2}\,\sqrt{4}\,t^3 + \frac{9}{4}\,t^4 + \frac{207}{64}\,\sqrt{4}\,t^5 + 21\,t^6 + \frac{150183}{4096}\,\sqrt{4}\,t^7 + \frac{137655}{512}\,t^8 + \frac{66893079}{131072}\,\sqrt{4}\,t^9 + 3978\,t^{10} + \frac{132735945771}{16777216}\,\sqrt{4}\,t^{11} + \frac{8385901667}{131072}\,t^{12} + \frac{70379121262905}{536870912}\,\sqrt{4}\,t^{13} + \frac{4345965}{4}\,t^{14} + \frac{78087826643607459}{34359738368}\,\sqrt{4}\,t^{15},\ -t^2]$$



A.7 Exempel 4

Kurvan $\left(t^3 - t^2, t^3 + t^2\right)$ har en singuläritet av ordning 1 i $t = 0$. Vi använder Maple för att omparametrisera kurvan:

```
> Reparametrize(t^3-t^2,t^3+t^2,t,0,10,0,true);
```

```
Elapsed Time: .110 s.
```

$$[-t^2,\ t^2 + 2\,t^3 + 3\,t^4 + \frac{21}{4}\,t^5 + 10\,t^6 + \frac{1287}{64}\,t^7 + 42\,t^8 + \frac{46189}{512}\,t^9 + 198\,t^{10}]$$

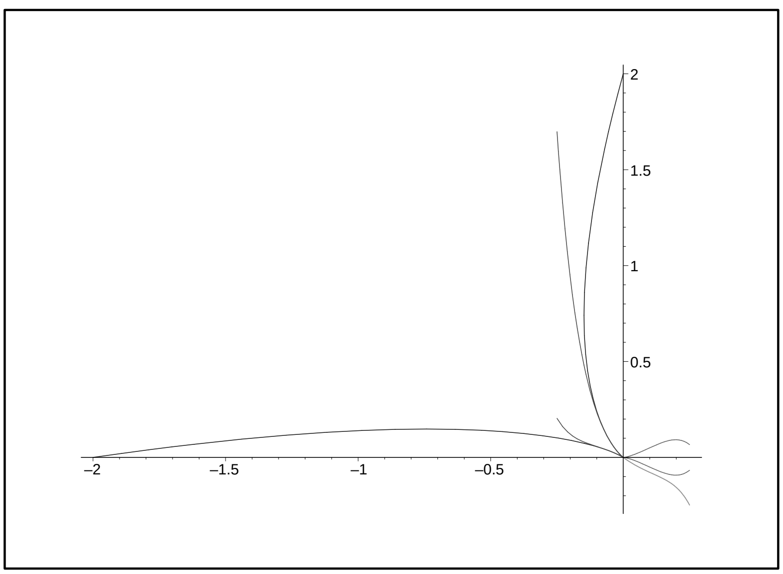
Som man kan se av omparametriseringen blir x-termen $-t^2$. Man kan be Maple att omparametrisera kurvan så att x-termen blir positiv. Detta medför givetvis att y-termen får komplexa koefficienter:

```
> Reparametrize(t^3-t^2,t^3+t^2,t,0,10,0,false);
```

```
Elapsed Time: .075 s.
```

$$[t^2,\ -t^2 - 2\,I\,t^3 + 3\,t^4 + \frac{21}{4}\,I\,t^5 - 10\,t^6 - \frac{1287}{64}\,I\,t^7 + 42\,t^8 + \frac{46189}{512}\,I\,t^9 - 198\,t^{10}]$$

Om vi plottar den ursprungliga funktionen och dess realvärda omparametriseringen tillsammans med realdelen och imaginärdelen av den komplexa omparametriseringen var för sig, får vi följande graf:

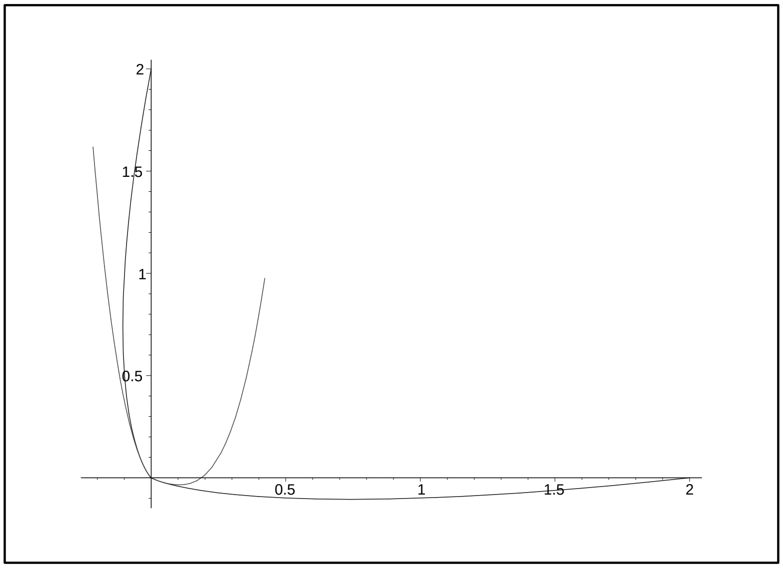


A.8 Exempel 5

Kurvan $\left(t^4 - t^3, t^4 + t^3\right)$ har en singuläritet av ordning 2 i $t = 0$. Den liknar mycket kurvan i exempel 4, men eftersom ordningen av x- och y-termerna är udda, kommer omparametriseringen att vara reell utan att x-termen måste negeras. Vi använder Maple för att omparametrisera kurvan:

```
> Reparametrize(t^4-t^3,t^4+t^3,t,0,10,0,false);
Elapsed Time: .040 s.
```

$$[t^3,\ -t^3+2\,t^4-\frac{8}{3}\,t^5+4\,t^6-\frac{520}{81}\,t^7+\frac{2618}{243}\,t^8-\frac{56}{3}\,t^9+\frac{217360}{6561}\,t^{10}]$$



Bilaga B

Semigrupper - källtexter till Maple

B.1 FindGCD

FindGCD-proceduren beräknar den största gemensamma delaren (greatest common divisor) mellan två heltal a och b. Den beräknar också de två heltalen A och B, sådana att gcd $= A\,a + B\,b$.

$FindGCD(a,\ b)$

- a Det första heltalet.
- b Det andra heltalet.

$FindGCD := \mathbf{proc}(a,\ b)$

local $a0,\ b0,\ m,\ n,\ r,\ GCD,\ BList,\ Position,\ A,\ B,\ switch;$

if not type(a, *integer*) **then**
ERROR(‘*a must be an integer*!‘, a)
end if;
if not type(b, *integer*) **then**
ERROR(‘*b must be an integer*!‘, b)
end if;
if $a < b$ **then**
$m := b\,;\ n := a\,;\ switch := true$
else
$m := a\,;\ n := b\,;\ switch := false$
end if;

$a0 := m\,;$
$b0 := n\,;$
$GCD := n\,;$
$r := m \bmod n\,;$
$Position := 1\,;$
while $r \neq 0$ **do**
  $BList_{Position} := -(m-r)/n\,;$
  $Position := Position + 1\,;$
  $m := n\,;$
  $n := r\,;$
  $GCD := r\,;$
  $r := m \bmod n$
end do;
$A := 0\,;$
$B := 1\,;$
while $1 < Position$ **do**
  $Position := Position - 1\,;$
  $r := B\,;$
  $B := A + BList_{Position} * B\,;$
  $A := r$
end do;
if *switch* **then**
  RETURN([GCD, B, A])
else
  RETURN([GCD, A, B])
end if
end proc

B.1.1 Exempel

Följande exempel beräknar den största gemensamma delaren för 12312334 och 239487192:

```
> FindGCD(12312334,239487192);
```

$$[2,\ -1065409,\ 54774]$$

Resultatet blir 2. Dessutom får vi:

$$2 = 12312334 \cdot (-1065409) + 239487192 \cdot 54774$$

B.2 FindGCDList

FindGCDList-proceduren beräknar den största gemensamma delaren för en lista med heltal (greatest common divisor). Den returnerar också en lista med heltal sådana att koefficienterna i den urspringliga heltalslistan multiplicerat med motsvarande koefficienter i den returnerade listan och sedan summerade blir lika med den största gemensamma delaren.

FindGCDList(*IntegerList*)

IntegerList En lista med heltal.

FindGCDList := **proc**(*IntegerList*)

local *NrIntegers*, *Position*, *GCD*, *Coefficients*, *l*;

if not type(*IntegerList*, *list*) **then**
ERROR(‘*IntegerList must be a list of integers*!‘, *IntegerList*)
end if;
NrIntegers := nops(*IntegerList*) ;
if $NrIntegers < 1$ **then**
ERROR(‘*IntegerList must be a list of at least one integer*!‘, *IntegerList*)
elif $NrIntegers = 1$ **then**
RETURN([$IntegerList_1$, [1]])
end if;
l := FindGCD($IntegerList_1$, $IntegerList_2$) ;
$GCD := l_1$;
$Coefficients := [l_2, l_3]$;
$Position := 3$;
while $Position \leq NrIntegers$ **do**
l := FindGCD(GCD, $IntegerList_{Position}$) ;
$GCD := l_1$;
$Coefficients :=$ [op($Coefficients * l_2$), l_3] ;
$Position := Position + 1$
end do;
RETURN([*GCD*, *Coefficients*])
end proc

B.2.1 Exempel

Följande exempel beräknar den största gemensamma delaren för 6, 9 och 20:

```
> FindGCDList([6,9,20]);
```

$$[1, [-7, 7, -1]]$$

Från svaret kan man också utläsa att

$$1 = 6 \cdot (-7) + 9 \cdot 7 + 20 \cdot (-1)$$

B.3 FindConductor

FindConductor-proceduren beräknar konduktören för en lista med heltal vars största gemensamma delare är 1. Sats (3.4) säger ju att en sådan finns (förutsatt att den största gemensamma delaren är 1).

FindConductor(*Generators*[, *PrintTime*])

Generators	En lista med heltal, vars största gemensamma delare måste vara 1.
PrintTime	En frivillig parameter. Om *true* så skrivs tiden som proceduren tagit ut på skärmen.

FindConductor := **proc**(*Generators*)

local *Conductor*, *l*, *MinValue*, *NrGenerators*, *g*, *ZMin*, *GCD*,
a, *b*, *c*, *d*, *e*, *StartTime*;

StartTime := time() ;
if not type(*Generators*, *list*) **then**
ERROR(‘*Generators must be a list of integers*!‘, *Generators*)
end if;
l := FindGCDList(*Generators*) ;
if $l_1 \neq 1$ **then** ERROR(
‘*The Generators must not have a common divisor greater than 1*!‘,
[*Generators*, l_1])
end if;
NrGenerators := nops(*Generators*) ;
MinValue := min(op(*Generators*)) ;

$ZMin$:= array(0..$MinValue - 1$) ;
$ZMin_0 := MinValue$;
for l **from** 1 **to** $MinValue - 1$ **do**
  $ZMin_l := 0$
end do ;
for l **to** $NrGenerators$ **do**
  $g := Generators_l$;
  if $g \neq MinValue$ **then**
    $GCD := \gcd(g,\ MinValue)$;
    $b := MinValue/GCD$;
    for e **from** 0 **to** $MinValue$ **do**
      if $e = MinValue$ **then**
        $c := 0$
      elif $ZMin_e \neq 0$ **then**
        $c := ZMin_e$
      else
        $c := -1$
      end if ;
      if $0 \leq c$ **then**
        for a **to** b **do**
          $c := c + g$;
          $d := c \bmod MinValue$;
          if $ZMin_d = 0$ **or** $c < ZMin_d$ **then**
            $ZMin_d := c$
          end if
        end do
      end if
    end do
  end if
end do;
$Conductor := ZMin_0$;
for a **to** $MinValue - 1$ **do**
  if $Conductor < ZMin_a$ **then**
    $Conductor := ZMin_a$
  end if
end do;
$Conductor := Conductor - MinValue + 1$;
if nargs = 1 **or** args_2 **then**
  printf("Elapsed Time: %0.3f s.\n", time() $-$ $StartTime$)
end if;
RETURN($Conductor$)
end proc

B.3.1 Exempel 1

I det första exemplet beräknar vi konduktören för $\langle 30, 42, 70, 105\rangle$. Notera att $30 = 2 \cdot 3 \cdot 5$, $42 = 2 \cdot 3 \cdot 7$, $70 = 2 \cdot 5 \cdot 7$ och $105 = 3 \cdot 5 \cdot 7$.

```
> FindConductor([2*3*5,2*3*7,2*5*7,3*5*7]);
Elapsed Time: 0.000 s.
```

$$384$$

Konduktören blir i detta exempel $384 = 2^7 \cdot 3$. Som man kan se i detta exempel verkar det inte finnas någon enkel självklar generalisering av formeln för konduktören av $\langle m, n\rangle$, då m och n är relativt prima ($c = (m-1)(n-1)$).

B.3.2 Exempel 2

Följande exempel visar att beräkningen av konduktören genomförs utan att motsvarande semigrupp genereras:

```
> FindConductor([2139,2398,3321]);
Elapsed Time: 11.316 s.
```

$$277188$$

Konduktören för $\langle 2139, 2398, 3321\rangle$ är alltså 277188. Beräkningen av motsvarande semigrupp skulle ta mycket längre tid.

B.4 FindSemiGroup

FindSemiGroup-proceduren genererar semigruppen från en lista med generatorer. Den returnerar konduktören och alla element fram till och med konduktören i semigruppen. Skulle inte generatorerna vara relativt prima (dvs de har en största gemensam delare större än 1), delar den först generatorerna med den största gemensamma delaren, skapar motsvarande semigrupp, multiplicerar elementen med den största gemensamma delaren och returnerar motsvarande "konduktör", den största gemensamma delaren, samt vilka element som ingår i semigruppe fram till och med "konduktören"[1].

FindSemiGroup(*Generators*[, *PrintTime*])

[1] Med "konduktören" i detta fall menas konduktören av semigruppen som genereras av generatorerna delat med den största gemensamma delaren, och därefter multiplceras denna konduktör med den största gemensamma delaren.

Generators	En lista med heltal.
PrintTime	En frivillig parameter. Om *true* så skrivs tiden som proceduren tagit ut på skärmen.

FindSemiGroup := **proc**(*Generators*)

local *Conductor*, *l*, *GCD*, *Generators2*, *MinValue*, *NrGenerators*, *g*,
ZMin, *GCD2*, *a*, *b*, *c*, *d*, *e*, *SemiGroup*, *StartTime*;

```
StartTime := time() ;
if not type(Generators, list) then
   ERROR(`Generators must be a list of integers!`, Generators)
end if;
l := FindGCDList(Generators) ;
GCD := l[1] ;
if GCD = 1 then
   Generators2 := Generators
else
   Generators2 := Generators/GCD
end if;
NrGenerators := nops(Generators2) ;
MinValue := min(op(Generators2)) ;
ZMin := array(0..MinValue − 1) ;
ZMin[0] := MinValue ;
for l from 1 to MinValue − 1 do
   ZMin[l] := 0
end do ;
for l to NrGenerators do
   g := Generators2[l] ;
   if g ≠ MinValue then
      GCD2 := gcd(g, MinValue) ;
      b := MinValue/GCD2 ;
      for e from 0 to MinValue do
         if e = MinValue then
            c := 0
         elif ZMin[e] ≠ 0 then
            c := ZMin[e]
         else
            c := −1
         end if ;
```

```
            if 0 ≤ c then
                for a to b do
                    c := c + g;
                    d := c mod MinValue;
                    if ZMin_d = 0 or c < ZMin_d then
                        ZMin_d := c
                    end if
                end do
            end if
        end do
    end if
end do;
Conductor := ZMin_0;
for a to MinValue − 1 do
    if Conductor < ZMin_a then
        Conductor := ZMin_a
    end if
end do;
Conductor := Conductor − MinValue + 1;
ZMin := array(1..Conductor);
for l to Conductor do
    ZMin_l := 0
end do;
for l to NrGenerators do
    g := Generators2_l;
    for a to Conductor + 1 do
        if Conductor < a then
            b := g
        elif ZMin_a ≠ 0 then
            b := ZMin_a + g
        else
            b := 0
        end if;
        if 0 < b then
            while b ≤ Conductor do
                ZMin_b := b;
                b := b + g
            end do
        end if
    end do
end do;
SemiGroup := {};
```

```
        for a to Conductor do
            if 0 < ZMin_a then
                SemiGroup := SemiGroup union {ZMin_a}
            end if
        end do;
        if GCD ≠ 1 then
            SemiGroup := GCD * SemiGroup
        end if;
        if nargs = 1 or args_2 then
            printf("Elapsed Time: %0.3f s.\n", time() − StartTime)
        end if;
        RETURN([Conductor * GCD, SemiGroup])
    end proc
```

B.4.1 Exempel 1

Det första exemplet beräknar $\langle 15, 10, 6\rangle$:

```
> FindSemiGroup([15,10,6]);
Elapsed Time: .010 s.
```

$$[30, \{6, 10, 12, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30\}]$$

Vi får att konduktören är 30 och att

$$\langle 15, 10, 6\rangle = \{6, 10, 12, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, \ldots\}$$

där "…" betyder "alla heltal som kommer därefter".

B.4.2 Exempel 2

Detta exempel beräknar $\langle 30, 20, 12\rangle$:

```
> FindSemiGroup([30,20,12]);
Elapsed Time: .030 s.
```

$$[60, 2\{6, 10, 12, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30\}]$$

Eftersom den största gemensamma delaren av 30, 20 och 12 är 2 (vilket även ses i resultatet) kan vi uttyda att

$$\langle 30, 20, 12\rangle = 2 \cdot \{6, 10, 12, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, \ldots\}$$

Multiplikationen av 2 med hela mängden skall betyda att elementen innanför mängen skall multipliceras med 2.

B.4.3 Exempel 3

I detta exempel beräknas $\langle 21, 93, 32\rangle$:

```
> FindSemiGroup([21,93,32]);
```

```
Elapsed Time: .010 s.
```

[332, {21, 32, 42, 53, 63, 64, 74, 84, 85, 93, 95, 96, 105, 106, 114, 116, 117, 125, 126, 127, 128, 135, 137, 138, 146, 147, 148, 149, 156, 157, 158, 159, 160, 167, 168, 169, 170, 177, 178, 179, 180, 181, 186, 188, 189, 190, 191, 192, 198, 199, 200, 201, 202, 207, 209, 210, 211, 212, 213, 218, 219, 220, 221, 222, 223, 224, 228, 230, 231, 232, 233, 234, 239, 240, 241, 242, 243, 244, 245, 249, 250, 251, 252, 253, 254, 255, 256, 260, 261, 262, 263, 264, 265, 266, 270, 271, 272, 273, 274, 275, 276, 277, 279, 281, 282, 283, 284, 285, 286, 287, 288, 291, 292, 293, 294, 295, 296, 297, 298, 300, 302, 303, 304, 305, 306, 307, 308, 309, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 323, 324, 325, 326, 327, 328, 329, 330, 332}]

B.4.4 Exempel 4

I detta exempel beräknar vi motsvarande semigrupp som användes i exempel 1 i beskrivningen av *FindConductor*-proceduren. Notera att det tar längre tid att beräkna semigruppen än konduktören.

```
> FindSemiGroup([2*3*5,2*3*7,2*5*7,3*5*7]);
```

```
Elapsed Time: .040 s.
```

[384, {30, 42, 60, 70, 72, 84, 90, 100, 102, 105, 112, 114, 120, 126, 130, 132, 135, 140, 142, 144, 147, 150, 154, 156, 160, 162, 165, 168, 170, 172, 174, 175, 177, 180, 182, 184, 186, 189, 190, 192, 195, 196, 198, 200, 202, 204, 205, 207, 210, 212, 214, 216, 217, 219, 220, 222, 224, 225, 226, 228, 230, 231, 232, 234, 235, 237, 238, 240, 242, 244, 245, 246, 247, 249, 250, 252, 254, 255, 256, 258, 259, 260, 261, 262, 264, 265, 266, 267, 268, 270, 272, 273, 274, 275, 276, 277, 279, 280, 282, 284, 285, 286, 287, 288, 289, 290, 291, 292, 294, 295, 296, 297, 298, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 312, 314, 315, 316, 317, 318, 319, 320, 321, 322, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 384}]

B.5 FindSemiGroupFromPolynomialRing

Proceduren *FindSemiGroupFromPolynomialRing* beräknar en serie generatorer för semigruppen som motsvarar polynomringen $R = \mathbb{C}[g_1, \ldots, g_n]$, där g_i är polynomen som anges i parameter 1. Semigruppen består av alla de heltal $n : \exists f \in R : n = \mathbf{o}(f)$.

FindSemiGroupFromPolynomialRing(*Polynomials*, *Variable* [, *PrintTime*[, *PrintFormulae*]])

Polynomials	En lista med polynom.
Variable	Namnet på variabeln.
PrintTime	En frivillig parameter. Om *true* så skrivs tiden som proceduren tagit ut på skärmen.
PrintFormulae	En frivillig parameter. Om *true* så skrivs för varje generator en explicita formel för hur motsvarande polynom genererats.

```
FindSemiGroupFromPolynomialRing := proc(PolynomialGenerators,
        Variable)

    localStartTime, GCD, Conductor, Polynomials, NrPolynomials,
        Generators, PolynomialsByOrder, Degrees, ExplicitNotation,
        FirstExplicitNotation, NrOriginalPolynomials, OriginalOrders,
        MaxExponent, MinOrder, HighestOrder, CalcExplicit, MaxTerm,
        MaxTerms, a, b, c, d, e, f, g, h, e1, e2, g1, g2, g3, d1, d2, d3, p;

        StartTime := time() ;
        CalcExplicit := 4 ≤ nargs and args_4 ;

        if not type(PolynomialGenerators, list) then
            ERROR(`PolynomialGenerators must be a list of polynomials!`,
                PolynomialGenerators)
        end if;

        NrOriginalPolynomials := nops(PolynomialGenerators);
        if NrOriginalPolynomials = 0 then
            ERROR(`List of polynomials cannot be empty!`)
        end if;

        f := PolynomialGenerators ;
        for a to NrOriginalPolynomials do
            g1 := f_a ;
            if not type(g1, polynom) then
                ERROR(`PolynomialGenerators must be a list of polynomials!`,
                    PolynomialGenerators)
            end if;
```

```
        ExplicitNotation_a := p_a ;
        OriginalOrders_a := ldegree(g1, Variable) ;
        Degrees_a := OriginalOrders_a
    end do;

    for a from 2 to NrOriginalPolynomials do
        for b to a − 1 do
            if Degrees_a < Degrees_b then
                c := Degrees_a ;
                Degrees_a := Degrees_b ;
                Degrees_b := c ;
                c := f_a ;
                f_a := f_b ;
                f_b := c ;
                c := ExplicitNotation_a ;
                ExplicitNotation_a := ExplicitNotation_b ;
                ExplicitNotation_b := c
            end if
        end do
    end do;

    MinOrder := Degrees_1 ;
    GCD := MinOrder ;
    Generators := [MinOrder] ;
    g1 := f_1 ;
    e := coeff(g1, Variable, GCD) ;
    Polynomials_1 := g1/e ;
    ExplicitNotation_1 := ExplicitNotation_1/e ;
    g := 1 ;

    for a from 2 to NrOriginalPolynomials do
        g1 := f_a ;
        b := ExplicitNotation_a ;
        for h to g do
            g2 := Polynomials_h ;
            d2 := ldegree(g2, Variable) ;
            e := coeff(g1, Variable, d2) ;
            if e ≠ 0 then
                g1 := g1 − e ∗ g2 ;
                b := b − e ∗ ExplicitNotation_h
            end if
        end do;
```

```
        if g1 ≠ 0 then
            d := ldegree(g1, Variable);
            Generators := [op(Generators), d];
            GCD := gcd(GCD, d);
            e := coeff(g1, Variable, d);
            g1 := g1/e;
            b := b/e;
            g := g + 1;
            Polynomials_g := g1;
            ExplicitNotation_g := b;

            for h to g − 1 do
                g2 := Polynomials_h;
                e := coeff(g2, Variable, d);
                if e ≠ 0 then
                    g2 := g2 − e * g1;
                    Polynomials_h := g2;
                    ExplicitNotation_h := ExplicitNotation_h − e * b
                end if
            end do

        end if
    end do;

    NrPolynomials := g;
    if GCD = 1 then
        Conductor := FindConductor(Generators, false);
        for c to NrOriginalPolynomials do
            MaxExponent_c := ceil((Conductor + 1)/OriginalOrders_c)
        end do
    else
        Conductor := ∞
    end if;

    HighestOrder := max(op(Generators));
    for a from MinOrder to HighestOrder do
        PolynomialsByOrder_a := 0
    end do;

    for a to NrPolynomials do
        f := Polynomials_a;
        d := ldegree(f, Variable);
        Degrees_a := d;
        PolynomialsByOrder_d := a;
        FirstExplicitNotation_d := ExplicitNotation_a
    end do;
```

```
a := 1 ;
while a ≤ NrPolynomials do
    g1 := Polynomials_a ;
    d1 := Degrees_a ;

    if g1 ≠ 0 and d1 ≤ Conductor then
        if CalcExplicit then
            e1 := ExplicitNotation_a
        end if;

        b := 1 ;
        while b ≤ a do
            if b = a then
                g2 := g1 ;
                d2 := d1
            else
                g2 := Polynomials_b ;
                d2 := Degrees_b
            end if;

            if g2 ≠ 0 and d2 ≤ Conductor then
                d := sort(simplify(expand(g1 * g2))) ;
                if CalcExplicit then
                    e2 := expand(e1 * ExplicitNotation_b)
                end if;

                d3 := d1 + d2 ;
                while d3 ≤ HighestOrder and d3 ≤ Conductor and
                    d ≠ 0 and PolynomialsByOrder_d3 ≠ 0 do

                    c := PolynomialsByOrder_d3 ;
                    g3 := Polynomials_c ;
                    if g3 = 0 then
                        ERROR(`Runtime Error`, d)
                    end if;

                    e := coeff(d, Variable, d3) ;
                    d := d − e * g3 ;
                    if CalcExplicit then
                        e2 := e2 − e * ExplicitNotation_c
                    end if;
                    d3 := ldegree(d, Variable)
                end do;
```

```
if d ≠ 0 and d3 ≤ Conductor then
    e := coeff(d, Variable, d3);
    d := d/e;
    if CalcExplicit then
        e2 := e2/e
    end if;

    Generators := [op(Generators), d3];
    GCD := gcd(GCD, d3);

    if GCD = 1 and Conductor = ∞ then
        Conductor := FindConductor(Generators, false);
        if Conductor < d3 then
            Conductor := d3
        end if;

        for c to NrOriginalPolynomials do
            MaxExponent_c := ceil((Conductor + 1)/OriginalOrders_c);
            MaxTerms_c := p_c^MaxExponent_c
        end do;

        MaxTerm := Variable^(Conductor+1);
        if Conductor < degree(d, Variable) then
            d := rem(d, MaxTerm, Variable);
            if CalcExplicit then
                for c toNrOriginalPolynomials do
                    if MaxExponent_c < degree(e2, p_c) then
                        e2 := expand(rem(e2, MaxTerms_c, p_c))
                    end if
                end do
            end if
        end if;

        for c to NrPolynomials do
            e := Polynomials_c;
            if e ≠ 0 and Conductor < degree(e, Variable) then
                Polynomials_c := rem(e, MaxTerm, Variable);
                if CalcExplicit then
                    if Polynomials_c = 0 then
                        ExplicitNotation_c := 0
                    end if
                else
```

$e := \mathit{ExplicitNotation}_c\,;$
for f **to** $\mathit{NrOriginalPolynomials}$ **do**
if $\mathit{MaxExponent}_f < \mathrm{degree}(e,\ p_f)$ **then**
$e := \mathrm{expand}(\mathrm{rem}(e,\ \mathit{MaxTerms}_f,\ p_f))$
end if
end do;
$\mathit{ExplicitNotation}_c := e$
end if
end if
end do

elif $\mathit{Conductor} < \mathrm{degree}(d,\ \mathit{Variable})$ **then**
$d := \mathrm{rem}(d,\ \mathit{MaxTerm},\ \mathit{Variable})\,;$
if $\mathit{CalcExplicit}$ **thenfor** c **to**
$\mathit{NrOriginalPolynomials}$ **do**
if $\mathit{MaxExponent}_c < \mathrm{degree}(e2,\ p_c)$ **then**
$e2 := \mathrm{expand}(\mathrm{rem}(e2,\ \mathit{MaxTerms}_c,\ p_c))$
end if
end do
end if
end if;

$\mathit{NrPolynomials} := \mathit{NrPolynomials} + 1\,;$
$\mathit{Polynomials}_{\mathit{NrPolynomials}} := d\,;$
$\mathit{Degrees}_{\mathit{NrPolynomials}} := d3\,;$

if $\mathit{CalcExplicit}$ **then**
$\mathit{ExplicitNotation}_{\mathit{NrPolynomials}} := e2\,;$
$\mathit{FirstExplicitNotation}_{d3} := e2$
end if;

if $\mathit{HighestOrder} < d3$ **then**
$\mathit{HighestOrder} := \mathit{HighestOrder} + 1\,;$
while $\mathit{HighestOrder} < d3$ **do**
$\mathit{PolynomialsByOrder}_{\mathit{HighestOrder}} := 0\,;$
$\mathit{HighestOrder} := \mathit{HighestOrder} + 1$
end do
end if;

```
                PolynomialsByOrder_{d3} := NrPolynomials ;
                h := NrPolynomials − 1 ;
                for c to h do
                  e := Polynomials_c ;
                  if e ≠ 0 then
                    f := coeff(e, Variable, d3) ;
                    if f ≠ 0 then
                      e := e − f ∗ d ;
                      NrPolynomials := NrPolynomials + 1 ;
                      g := Degrees_c ;
                      Polynomials_{NrPolynomials} := e ;
                      Degrees_{NrPolynomials} := g ;
                      PolynomialsByOrder_g := NrPolynomials ;

                      if CalcExplicit then
                        e := ExplicitNotation_c − f ∗ e2 ;
                        ExplicitNotation_{NrPolynomials} := e
                      end if;

                      Polynomials_c := 0
                    end if
                  end if
                end do

              end if
            end if;

            b := b + 1
          end do
        end if;

        a := a + 1
      end do;

      if Conductor = ∞ then
        ERROR(‘No upper bound found.‘, PolynomialGenerators)
      end if;
```

$Generators := \mathrm{sort}(Generators)\,;$
$a := \{\}\,;$
$b := [\,]\,;$
for c **to** $\mathrm{nops}(Generators)$ **do**
  $d := Generators_c\,;$
  if $d \leq Conductor$ **and not** $\mathrm{member}(d,\, a)$ **then**
    $b := [\mathrm{op}(b),\, d]\,;$
    $e := d\,;$
    $f := \{\}\,;$
    while $e \leq Conductor$ **do**
      $f := f$ union $\{e\}\,;$
      for h **to** $\mathrm{nops}(a)$ **do**
        $g := a_h + e\,;$
        if $g \leq Conductor$ **then** $f := f$ union $\{g\}$ **end if**
      end do;
      $e := e + d$
    end do;
    $a := a$ union f
  end if
end do;

$Generators := b\,;$

if $CalcExplicit$ **then**
  for c **to** $NrOriginalPolynomials$ **do**
    $MaxExponent_c := \mathrm{ceil}((Conductor + 1)/OriginalOrders_c);$
    $MaxTerms_c := {p_c}^{MaxExponent_c}$
  end do;
  $MaxTerm := Variable^{(Conductor+1)}\,;$

  for a **to** $\mathrm{nops}(Generators)$ **do**
    $b := Generators_a\,;$
    $c := FirstExplicitNotation_b\,;$
    for e **to** $NrOriginalPolynomials$ **do**
      $c := \mathrm{expand}(\mathrm{rem}(c,\, MaxTerms_e,\, p_e))$
    end do;

$d := \text{convert}(c,\ \textit{list})\,;$
if $1 < \text{nops}(d)$ **thenfor** e **to** $\text{nops}(d)$ **do**
$\quad f := d_e\,;$
$\quad$ **for** g **to** *NrOriginalPolynomials* **do**
$\quad\quad f := \text{subs}(p_g = \textit{PolynomialGenerators}_g,\ f)$
$\quad$ **end do**;
$\quad$ **if** $b < \text{ldegree}(f)$ **then** $c := c - d_e$ **end if**
end do
end if;

$d := c\,;$
for g **to** *NrOriginalPolynomials* **do**
$\quad d := \text{subs}(p_g = \textit{PolynomialGenerators}_g,\ d)$
end do;

$d := \text{sort}(\text{expand}(d))\,;$
$e := \text{lcm}(\text{denom}(c),\ \text{denom}(d))\,;$
$c := \text{sort}(e * c)\,;$
$d := \text{sort}(e * d)\,;$
$\text{print}(c = d)$
end do
end if;

if $\text{nargs} < 3$ **or** args_3 **then**
$\quad$ printf("Elapsed Time: %0.3f s.\n", time() $-$ *StartTime*)
end if;

RETURN(*Generators*)
end proc

B.5.1 Exempel 1

I följande exempel beräknar vi den numeriska semigruppen G_1 motsvarande $\mathbb{C}[t^4 + t^5, t^6 + t^7]$:

```
> FindSemiGroupFromPolynomialRing([t^4+t^5,t^6+t^7],t,
true,true);
```

$$p_1 = t^5 + t^4$$

$$p_2 = t^7 + t^6$$

$$p_1{}^3 - p_2{}^2 = t^{15} + 2\,t^{14} + t^{13}$$

```
Elapsed Time: .101 s.
```

$$[4, 6, 13]$$

```
> FindSemiGroup([4,6,13]);

Elapsed Time: 0.000 s.
```

$$[16, \{4, 6, 8, 10, 12, 13, 14, 16\}]$$

Vi fick alltså att $G_1 = \langle 4, 6, 13\rangle = \{4, 6, 8, 10, 12, 13, 14, 16, \ldots\}$. Konduktören för semigruppen är 16. Vi fick dessutom veta att 13 kommer från ${p_1}^3 - {p_2}^2 = t^{15} + 2\,t^{14} + t^{13}$, där $p_1 = t^5 + t^4$ och $p_2 = t^7 + t^6$.

Vi kan också jämföra detta resultat med den numeriska semigruppen som motsvarar polynomringen vi får om vi först gör en omparametrisering av kurvan i punkten $t = 0$:

```
> Reparametrize(t^4+t^5,t^6+t^7,t,0,15,0,false);

Elapsed Time: .171 s.
```

$$[t^4, t^6 + \frac{1}{2}\,t^7 + \frac{1}{2}\,t^8 + \frac{39}{64}\,t^9 + \frac{105}{128}\,t^{10} + \frac{4807}{4096}\,t^{11} + \frac{7}{4}\,t^{12} + \frac{352495}{131072}\,t^{13} + \frac{138567}{32768}\,t^{14} + \frac{113591595}{16777216}\,t^{15}]$$

```
> FindSemiGroupFromPolynomialRing([t^4,t^6 + 1/2*t^7 +
1/2*t^8 + 39/64*t^9 + 105/128*t^10 + 4807/4096*t^11 +
7/4*t^12 + 352495/131072*t^13 + 138567/32768*t^14 +
113591595/16777216*t^15],t,true,true);
```

$$p_1 = t^4$$

$$\begin{aligned} 16777216\,p_2 &= 113591595\,t^{15} + 70946304\,t^{14} + 45119360\,t^{13} + \\ &+ 29360128\,t^{12} + 19689472\,t^{11} + 13762560\,t^{10} + \\ &+ 10223616\,t^9 + 8388608\,t^8 + 8388608\,t^7 + 16777216\,t^6 \end{aligned}$$

$$\begin{aligned} &-281474976710656\,{p_1}^3 + 281474976710656\,{p_2}^2 = \\ &\quad 12903050454644025\,t^{30} + 16117807661429760\,t^{29} + \\ &\quad 15283738186818816\,t^{28} + 13072231199539200\,t^{27} + \\ &\quad 10674858838319104\,t^{26} + 8569833185345536\,t^{25} + \\ &\quad 6914209094303744\,t^{24} + 5754492897198080\,t^{23} + \\ &\quad 5214414567374848\,t^{22} + 6901048593612800\,t^{21} + \\ &\quad 4222124650659840\,t^{20} + 2618276488151040\,t^{19} + \\ &\quad 1650916709105664\,t^{18} + 1063090305105920\,t^{17} + \\ &\quad 703687441776640\,t^{16} + 483785116221440\,t^{15} + \\ &\quad 351843720888320\,t^{14} + 281474976710656\,t^{13} \end{aligned}$$

```
Elapsed Time: .101 s.
```

$$[4, 6, 13]$$

Från detta ser vi att inte bara är den numeriska semigruppen oförändrad vid omparametriseringen, utan vi kan även notera att generatorn 13 genereras på samma sätt:

$$\mathbf{o}({p_1}^3 - {p_2}^2) = 13$$

B.5.2 Exempel 2

Följande exempel är mer beräkningsintensiv. Vi skall beräkna den numeriska semigruppen G_2 motsvarande $\mathbb{C}[t^8 + t^{11}, t^{12} + t^{13}]$. För att se skillnaden i tidsåtgång mellan att endast beräkna generatorerna till semigruppen och att dessutom beräkna hur man beräknar motsvarande polynom gör vi två exekveringar enligt följande:

```
> FindSemiGroupFromPolynomialRing([t^8+t^11,t^12+t^13],t);

Elapsed Time: 4.386 s.
```

$$[8, 12, 25]$$

```
> FindSemiGroupFromPolynomialRing([t^8+t^11,t^12+t^13],t,
true,true);
```

$$p_1 = t^{11} + t^8$$

$$p_2 = t^{13} + t^{12}$$

$$-{p_1}^3 + {p_2}^2 = -t^{33} - 3\,t^{30} - 3\,t^{27} + t^{26} + 2\,t^{25}$$

```
Elapsed Time: 115.697 s.
```

$$[8, 12, 25]$$

```
> FindSemiGroup([8,12,25]);

Elapsed Time: 0.000 s.
```

[80, {8, 12, 16, 20, 24, 25, 28, 32, 33, 36, 37, 40, 41, 44, 45, 48, 49, 50, 52, 53, 56, 57, 58, 60, 61, 62, 64, 65, 66, 68, 69, 70, 72, 73, 74, 75, 76, 77, 78, 80}]

Vi fick alltså att $G_2 = \langle 8, 12, 25 \rangle$. Konduktören för semigruppen är 80. Vi fick dessutom veta att 25 kommer från $-{p_1}^3 + {p_2}^2 = -t^{33} - 3\,t^{30} - 3\,t^{27} + t^{26} + 2\,t^{25}$, där $p_1 = t^{11} + t^8$ och $p_2 = t^{13} + t^{12}$.

För att förstå varför detta exempel är så beräkningsintensivt i jämförelse med exempel 1 betraktar vi förhållandet mellan konduktörerna och den lägsta av generatorerna i varje exempel.

I Exempel 1 är den lägsta generatorn 4 och konduktören 16. Dvs den största potens av denna generator som kan förekomma i algoritmen är 4 i detta exempel.

I detta exempel är den lägsta generatorn 8 men konduktören är 80. Detta betyder att den lägsta potensen av denna generator som förekommer i algoritmen är 10. Antalet kombinationer av generatorerna är i detta exempel mycket större, varför algoritmen måste generera betydligt fler polynom innan den är klar.

Dessutom växer polyomen i takt med att större potenser förekommer i dem. Detta gör polynommultiplikationer kostbara m.a.p. tidsåtgång. Att denna faktor är betydande ser man i detta senare exempel. Här beräknar vi semigruppen två gånger, först bara generatorerna, och sedan även hur dessa generatorer kan fås från de ursprungliga polynomen. Eftersom komplexiteten är densamma i de två anropen ser vi att polynomaritmetiken i det senare fallet gjorde att rutinen tog 26 ggr längre tid.

Även i detta exempel undersöker vi vad som händer om vi först gör en omparametrisering av kurvan:

```
> Reparametrize(t^8+t^11,t^12+t^13,t,0,20,2,false);
```

```
Elapsed Time: 3.335 s.
```

$$[t^8,\, t^{12} + t^{13} - \frac{3}{2}\,t^{15} - \frac{13}{8}\,t^{16} + \frac{39}{16}\,t^{18} + \frac{351}{128}\,t^{19}]$$

```
> FindSemiGroupFromPolynomialRing([t^8, t^12 +
t^13-3/2*t^15-13/8*t^16+39/16*t^18+351/128*t^19],
t,true,true);
```

$$p_1 = t^8$$

$$\begin{aligned} 128\,p_2 &= 351\,t^{19} + 312\,t^{18} - 208\,t^{16} - 192\,t^{15} + 128\,t^{13} + \\ &+ 128\,t^{12} \end{aligned}$$

$$\begin{aligned}
-16384\,{p_1}^3 + 16384\,{p_2}^2 &= 123201\,t^{38} + 219024\,t^{37} + 97344\,t^{36} - \\
&- 146016\,t^{35} - 264576\,t^{34} - 119808\,t^{33} + \\
&+ 133120\,t^{32} + 249600\,t^{31} + 116736\,t^{30} - \\
&- 53248\,t^{29} - 102400\,t^{28} - 49152\,t^{27} + \\
&+ 16384\,t^{26} + 32768\,t^{25}
\end{aligned}$$

```
Elapsed Time: 206.127 s.
```

[8, 12, 25]

Även i detta exempel förblir semigruppen oförändrad. Dessutom fås generatorerna på samma sätt: $\mathbf{o}({p_1}^3 - {p_2}^2) = 25$.

B.5.3 Exempel 3

Följande är ett enkelt exempel på att vi kan använda oss av fler än två polynom som indata. Vi skall beräkna den numeriska semigruppen G_3 motsvarande $\mathbb{C}[t^5, t^7, t^{12} + t^{13}]$:

```
> FindSemiGroupFromPolynomialRing([t^5,t^7,t^12+t^13],t,
true,true);
```

$$p_1 = t^5$$

$$p_2 = t^7$$

$$-p_1\,p_2 + p_3 = t^{13}$$

```
Elapsed Time: .010 s.
```

[5, 7, 13]

```
> FindSemiGroup([5,7,13]);
```

```
Elapsed Time: 0.000 s.
```

[17, {5, 7, 10, 12, 13, 14, 15, 17}]

Vi fick alltså att $G_3 = \langle 5, 7, 13\rangle$={5, 7, 10, 12, 13, 14, 15, 17}. Konduktören för semigruppen är 17. Vi fick dessutom veta att 13 kommer från $-p_1\,p_2 + p_3 = t^{13}$, där $p_1 = t^5$, $p_2 = t^7$ och $p_3 = t^{12} + t^{13}$.

B.5.4 Exempel 4

Här är ett annat enkelt litet exempel som visar att kedjan för att hitta generatorerna kan vara litet längre. Vi skall beräkna den numeriska semigruppen G_4 motsvarande $\mathbb{C}[t^4, t^6 + t^8 + t^{11}]$.

```
> FindSemiGroupFromPolynomialRing([t^4,t^6+t^8+t^11],
t,true,true);
```

$$p_1 = t^4$$

$$p_2 = t^{11} + t^8 + t^6$$

$$p_1{}^4 - p_1{}^3 - 2\,p_1{}^2\,p_2 + p_2{}^2 = t^{22} + 2\,t^{17}$$

```
Elapsed Time: .010 s.
```

$$[4,\ 6,\ 17]$$

```
> FindSemiGroup([4,6,17]);
```

```
Elapsed Time: 0.000 s.
```

$$[20,\ \{4,\ 6,\ 8,\ 10,\ 12,\ 14,\ 16,\ 17,\ 18,\ 20\}]$$

Vi får att $G_4 = \langle 4, 6, 17\rangle$={4, 6, 8, 10, 12, 14, 16, 17, 18, 20}. Konduktören för semigruppen är 20. Vi fick dessutom veta att generatorn 17 kommer från $p_1{}^4 - p_1{}^3 - 2\,p_1{}^2\,p_2 + p_2{}^2 = t^{22} + 2\,t^{17}$, där $p_1 = t^4$ och $p_2 = t^{11} + t^8 + t^6$.

B.5.5 Exempel 5

Ett annat trivialt exempel: Här beräknas den numeriska semigruppen G_5 motsvarande $\mathbb{C}[t^2, t^4 + t^6 + t^{10} + t^{13}]$.

```
> FindSemiGroupFromPolynomialRing([t^2,t^4+t^6+t^10+t^13],
t,true,true);
```

$$p_1 = t^2$$

$$-p_1{}^5 - p_1{}^3 - p_1{}^2 + p_2 = t^{13}$$

```
Elapsed Time: .010 s.
```

$$[2,\ 13]$$

```
> FindSemiGroup([2,13]);
```

```
Elapsed Time: 0.000 s.
```

$$[12, \{2, 4, 6, 8, 10, 12\}]$$

Vi fick att $G_5 = \langle 2, 13\rangle$={2, 4, 6, 8, 10, 12}. Konduktören för semigruppen är 12 (dvs mindre än den större generatorn). Vi fick dessutom veta att 13 kommer från $-{p_1}^5 - {p_1}^3 - {p_1}^2 + p_2 = t^{13}$, där $p_1 = t^2$ och $p_2 = t^4 + t^6 + t^{10} + t^{13}$.

B.5.6 Exempel 6

Här kommer ett litet besvärligare exempel: Här beräknas den numeriska semigruppen G_6 motsvarande $\mathbb{C}[t^2 + t^5, t^4 + t^6 + t^{10} + t^{13}]$.

```
> FindSemiGroupFromPolynomialRing([t^2+t^5,
t^4+t^6+t^10+t^13],t,true,true);
```

$$p_1 = t^5 + t^2$$

$${p_1}^3 + {p_1}^2 - p_2 = t^{15} - t^{13} + 3\,t^{12} + 3\,t^9 + 2\,t^7$$

```
Elapsed Time: .010 s.
```

$$[2, 7]$$

```
> FindSemiGroup([2,7]);
```

```
Elapsed Time: 0.000 s.
```

$$[6, \{2, 4, 6\}]$$

Vi fick att $G_6 = \langle 2, 7\rangle$={2, 4, 6}. Konduktören för semigruppen är 6 (dvs mindre än den större generatorn). Vi fick dessutom veta att 7 kommer från ${p_1}^3 + {p_1}^2 - p_2 = t^{15} - t^{13} + 3\,t^{12} + 3\,t^9 + 2\,t^7$, där $p_1 = t^2 + t^5$ och $p_2 = t^4 + t^6 + t^{10} + t^{13}$.

För skojs skull gör vi även här en omparametrisering för att se dess motsvarande semigrupp, och hur generatorerna fås:

```
> Reparametrize(t^2+t^5,t^4+t^6+t^10+t^13,t,0,10,0,false);
```

```
Elapsed Time: .010 s.
```

$$[t^2,\ t^4 + t^6 - 2\,t^7 - 3\,t^9 + 7\,t^{10}]$$

```
> FindSemiGroupFromPolynomialRing([t^2,t^4+t^6-2*t^7
-3*t^9+7*t^10],t,true,true);
```

$$p_1 = t^2$$

$$p_1{}^3 + p_1{}^2 - p_2 = -7\,t^{10} + 3\,t^9 + 2\,t^7$$

```
Elapsed Time: 0.000 s.
```

$$[2,\ 7]$$

Återigen ser vi att semigruppen förblir oförändrad vid omparametrisering. Även generatorerna fås på samma sätt: $\mathbf{o}(p_1{}^3 + p_1{}^2 - p_2) = 7$.

Bilaga C

Implicit notation - källtexter till Maple

C.1 FindImplicitNotation

FindImplicitNotation-proceduren beräknar en kurva F(x,y)=0 sådan att F(Xp,Yp)=0. F(x,y) har minimal ordning i den variabel som används för att definiera Xp och Yp. Funktionen tar också en maximal ordning som används i sökalgoritmen, för att begränsa sökningen.

FindImplicitNotation(X_p, Y_p, *Variable*, *MaxOrder*[, *PrintTime*])

X_p	Parametriseringen av x-koordinaten.
Y_p	Parametriseringen av y-koordinaten.
Variable	Vilken variabel som använts vid parametriseringen.
MaxOrder	Maximal ordning vid sökning efter medlemmar i $\langle X_p , Y_p \rangle$.
PrintTime	Valfri parameter. Styr om tidsåtgången ska skrivas ut.

$FindImplicitNotation := \textbf{proc}(Xp,\ Yp,\ Variable,\ MaxOrder)$
  $\textbf{local} List,\ Pos,\ Count,\ Degree,\ Degrees,\ MaxDeg,\ Explicit,\ StartTime,$
  $CurrentMaxOrder,\ PrintTime,\ Solution,\ i,\ j,\ k,\ f,\ g,\ p,\ x,\ y,\ a,\ b;$
    $StartTime := \mathrm{time}()\,;$
    $PrintTime := \mathrm{nargs} < 5\ \textbf{or}\ \mathrm{args}_5\,;$
    $CurrentMaxOrder := MaxOrder\,;$
    $Solution := 0\,;$
    $f := Xp\,;$
    $g := Yp\,;$

if $\mathrm{ldegree}(f,\ \mathit{Variable}) = \mathrm{ldegree}(g,\ \mathit{Variable})$ **then**
    $g := g * \mathrm{tcoeff}(f,\ \mathit{Variable}) - f * \mathrm{tcoeff}(g,\ \mathit{Variable})$
end if;
$\mathit{List}_0 := f$;
$\mathit{List}_1 := g$;
$\mathit{Explicit}_0 := x$;
$\mathit{Explicit}_1 := y$;
$\mathit{MaxDeg} := \mathrm{ldegree}(f,\ \mathit{Variable})$;
$\mathit{Degree}_0 := \mathit{MaxDeg}$;
for i **from** 0 **to** $\mathit{MaxDeg} - 1$ **do** $\mathit{Degrees}_i := -1$ **end do** ;
$\mathit{Degrees}_{\mathit{MaxDeg}} := 0$;
$j := \mathrm{ldegree}(g,\ \mathit{Variable})$;
if $\mathit{MaxDeg} < j$ **then**
    if $\mathit{MaxDeg} + 1 < j$ **then**
        for i **from** $\mathit{MaxDeg} + 1$ **to** $j - 1$ **do** $\mathit{Degrees}_i := -1$ **end do**
    end if;
    $\mathit{MaxDeg} := j$
end if;
$\mathit{Degrees}_j := 1$;
$\mathit{Degree}_1 := j$;
$\mathit{Pos} := 0$;
$\mathit{Count} := 2$;
while $\mathit{Pos} < \mathit{Count}$ **do**
    for i **from** 0 **to** Pos **do**
        if $\mathit{Degree}_{\mathit{Pos}} + \mathit{Degree}_i \leq \mathit{CurrentMaxOrder}$ **then**
            $f := \mathrm{expand}(\mathit{List}_{\mathit{Pos}} * \mathit{List}_i)$;
            $p := \mathrm{expand}(\mathit{Explicit}_{\mathit{Pos}} * \mathit{Explicit}_i)$;
            $j := \mathrm{ldegree}(f,\ \mathit{Variable})$;
            $k := 0$;
            while $j \leq \mathit{MaxDeg}$ **and** $f \neq 0$ **do**
                $k := \mathit{Degrees}_j$;
                if $0 \leq k$ **then**
                    $g := \mathit{List}_k$;
                    $a := \mathrm{tcoeff}(g,\ \mathit{Variable})$;
                    $b := \mathrm{tcoeff}(f,\ \mathit{Variable})$;
                    $f := f * a - g * b$;
                    $p := p * a - \mathit{Explicit}_k * b$;
                    if $f \neq 0$ **then** $j := \mathrm{ldegree}(f,\ \mathit{Variable})$ **end if**

```
            else
                List_Count := f ;
                Explicit_Count := p ;
                Degree_Count := j ;
                Degrees_j := Count ;
                Count := Count + 1 ;
                f := 0
            end if
        end do;
        if f ≠ 0 and MaxDeg < j then
            while MaxDeg < j − 1 do MaxDeg := MaxDeg + 1 ; Degrees_MaxDeg := −1
            end do;
            List_Count := f ;
            Explicit_Count := p ;
            Degree_Count := j ;
            Degrees_j := Count ;
            MaxDeg := j ;
            Count := Count + 1
        elif 0 ≤ k and p ≠ 0 then
            j := degree(p) ;
            if j < CurrentMaxOrder then CurrentMaxOrder := j ; Solution := p end if
        end if
      end if
    end do;
    Pos := Pos + 1
  end do;
  if Solution = 0 thenERROR(
    ‘Implicit function not found. Please try a higher Maximum Order.‘, MaxOrder)
  end if;
  if PrintTime then printf(“Elapsed Time: %0.3f s.\n”, time() − StartTime) end if;
  a := [coeffs(Solution)] ;
  b := nops(a) ;
  j := op(1, a) ;
  for i from 2 to b do j := gcd(j, op(i, a)) end do ;
  Solution := Solution/j ;
  RETURN(sort(Solution) = 0)
end proc
```

C.1.1 Exempel 1

Följande exempel beräknar den implicita funktionen till kurvan $C(t) = (t^2, t^3)$:

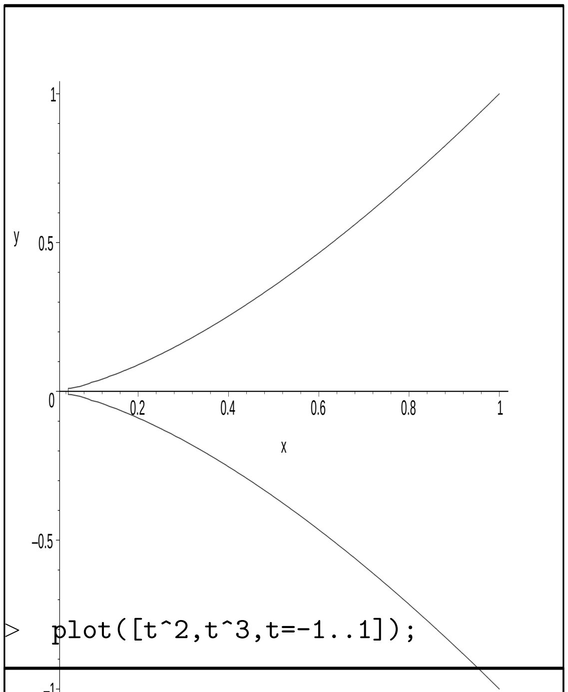
```
> FindImplicitNotation(t^2,t^3,t,30,true);
```

```
Elapsed Time: 0.000 s.
```

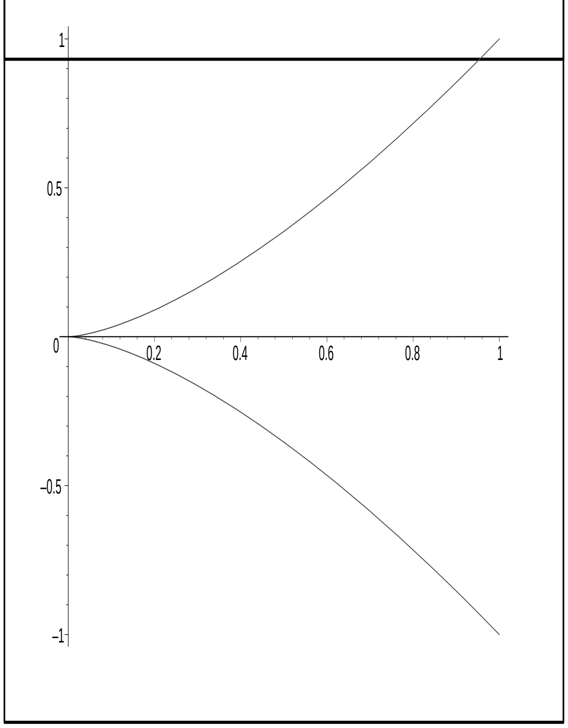
$$x^3 - y^2 = 0$$

Kurvan $C(t)$ motsvaras altså av $x^3 - y^2 = 0$.

```
> implicitplot(x^3-y^2 = 0,x=-1..1,y=-1..1,grid=[100,100]);
```




```
> plot([t^2,t^3,t=-1..1]);
```



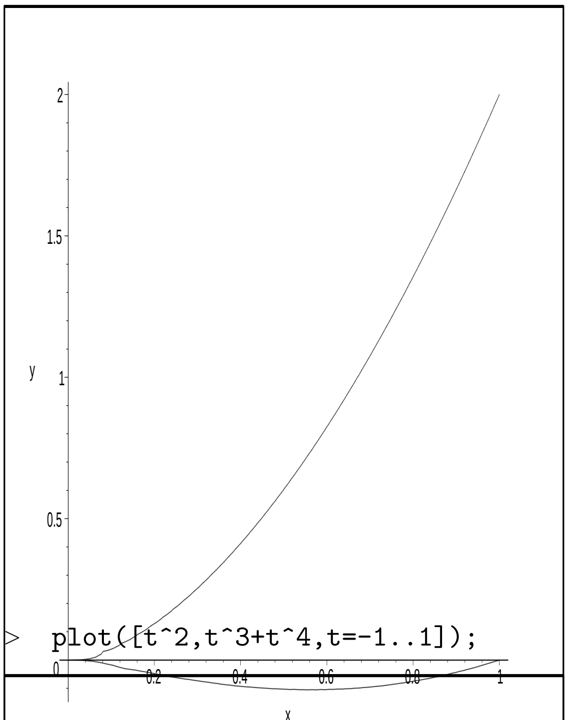
C.1.2 Exempel 2

```
> FindImplicitNotation(t^2,t^3+t^4,t,30,true);
```

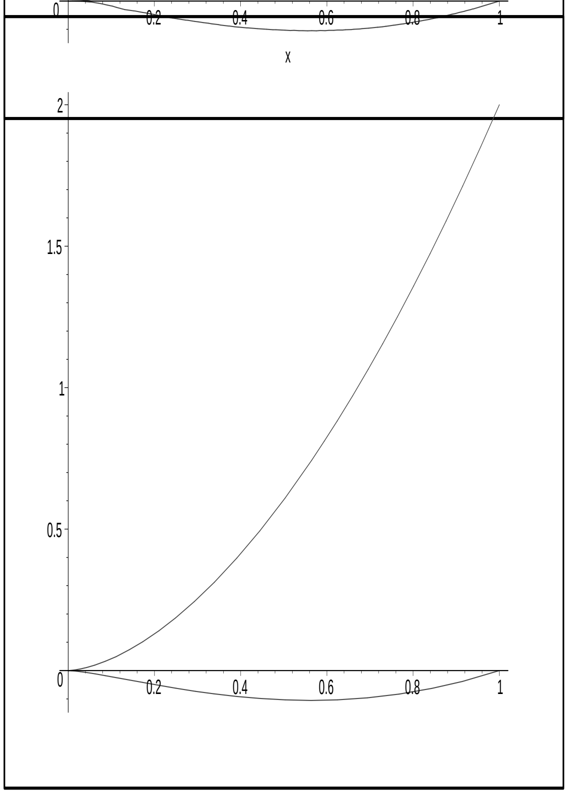
```
Elapsed Time: 0.000 s.
```

$$-x^4 + x^3 + 2\,x^2\,y - y^2 = 0$$

```
> implicitplot(-x^4+2*y*x^2+x^3-y^2=0,x=-1..1,y=-1..2,grid=[100,100]);
```




```
> plot([t^2,t^3+t^4,t=-1..1]);
```



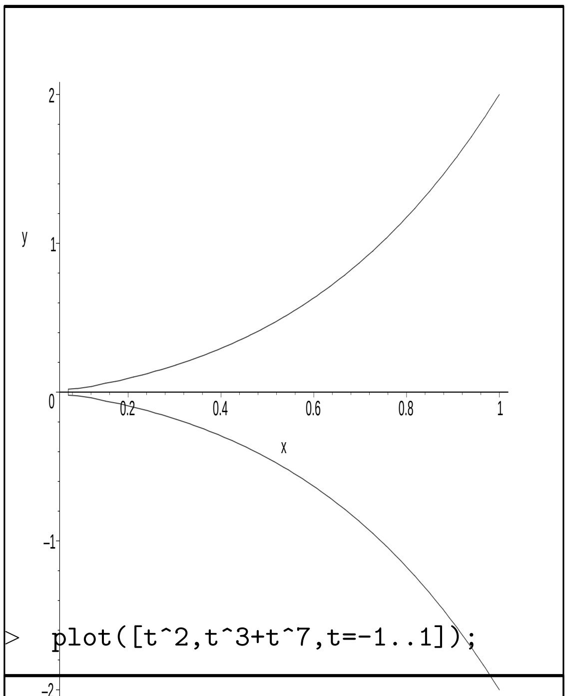
C.1.3 Exempel 3

```
> FindImplicitNotation(t^2,t^3+t^7,t,30,true);
```

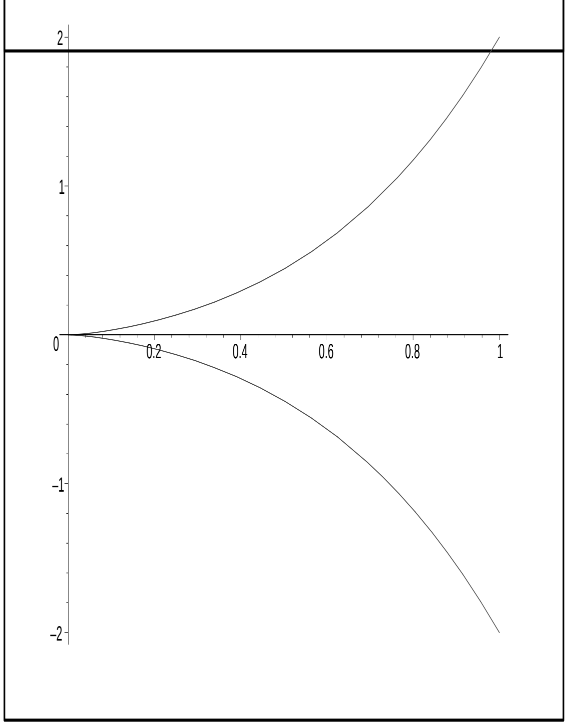
```
Elapsed Time: 0.000 s.
```

$$-x^7 - 2\,x^5 - x^3 + y^2 = 0$$

```
> implicitplot(x^7+2*x^5+x^3-y^2=0,x=-1..2,y=-2..2,grid=[100,100]);
```




```
> plot([t^2,t^3+t^7,t=-1..1]);
```



C.1.4 Exempel 4

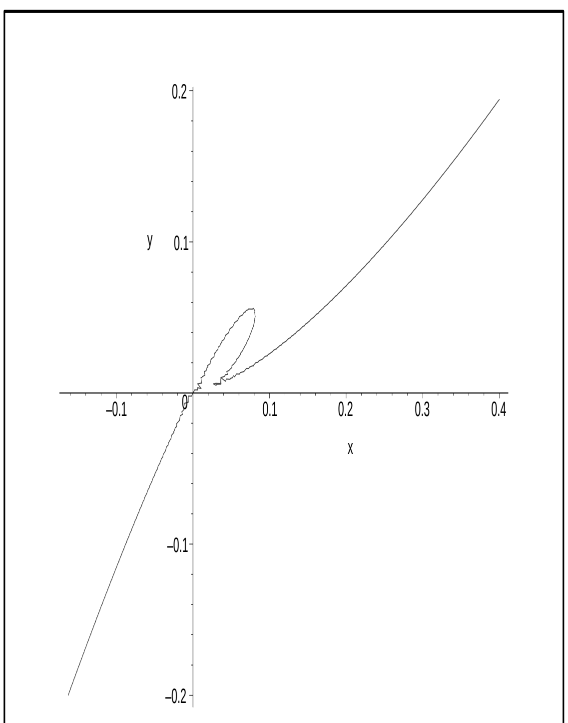
Detta exempel motsvarar exempel 1 för
FindSemiGroupFromPolynomialRing.

```
> FindImplicitNotation(t^4+t^5,t^6+t^7,t,40,true);
```

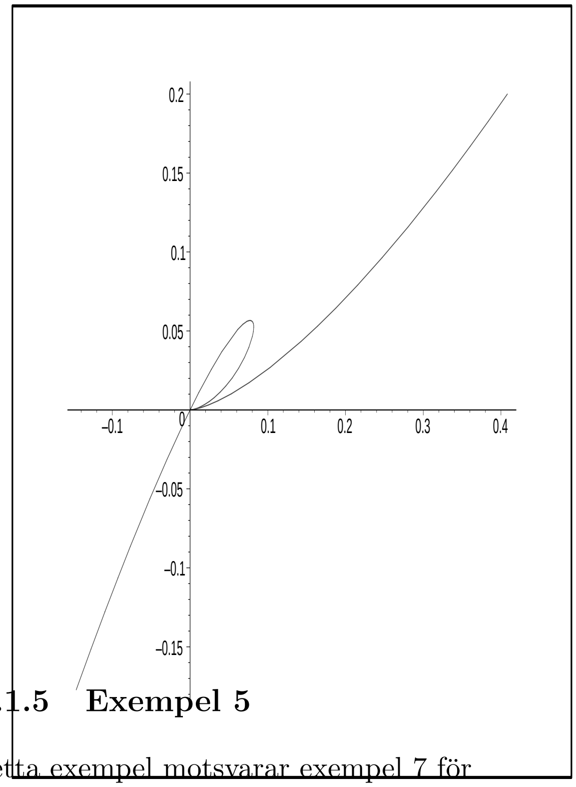
```
Elapsed Time: .020 s.
```

$$-x^7 + 2\,y^2\,x^4 + y^5 - y^4\,x = 0$$

```
> implicitplot(x^7-2*y^2*x^4-y^5+y^4*x=0,x=-0.2..0.4,y=-0.2..0.2,grid=[
100,100]);
```




```
> plot([t^4+t^5,t^6+t^7,t=-1.1..0.7]);
```



C.1.5 Exempel 5

Detta exempel motsvarar exempel 7 för FindSemiGroupFromPolynomialRing.

```
> FindImplicitNotation(t^8,t^12+t^14+t^15,t,150,true);
```

```
Elapsed Time: .220 s.
```

$$x^{15}+8\,y\,x^{13}-21\,x^{14}+20\,y^2\,x^{11}-16\,y\,x^{12}-6\,x^{13}+24\,y^3\,x^9-36\,y^2\,x^{10}+8\,y\,x^{11}-x^{12}+26\,y^4\,x^7-16\,y^3\,x^8+4\,y^2\,x^9+8\,y^5\,x^5-6\,y^4\,x^6+4\,y^6\,x^3-y^8=0$$

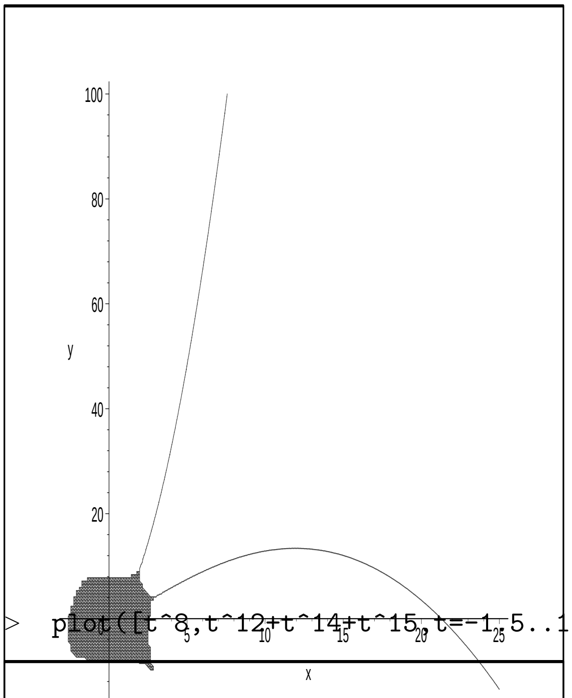
```
> StartTime:=time();sort(resultant(x-t^8,y-t^12-t^14-t^15,t));time()-St
artTime;
```

$$StartTime := 25.866$$

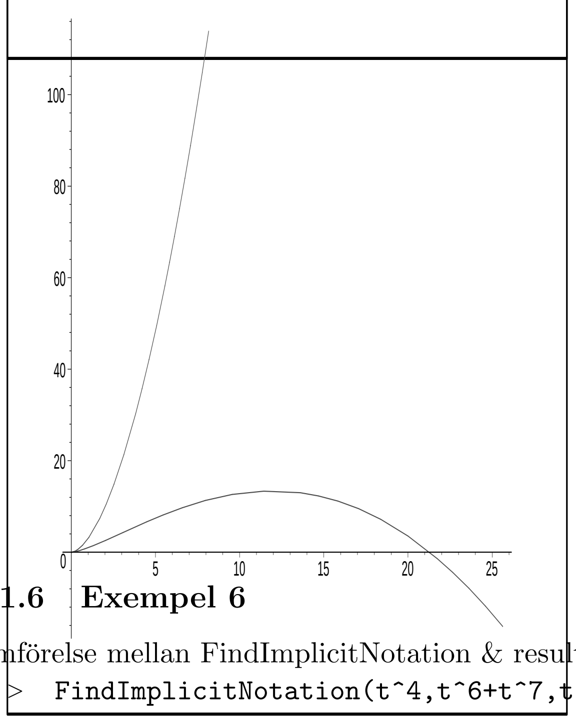
$$x^{15}-21\,x^{14}+8\,x^{13}\,y-6\,x^{13}-16\,x^{12}\,y+20\,x^{11}\,y^2-x^{12}+8\,x^{11}\,y-36\,x^{10}\,y^2+24\,x^9\,y^3+4\,x^9\,y^2-16\,x^8\,y^3+26\,x^7\,y^4-6\,x^6\,y^4+8\,x^5\,y^5+4\,x^3\,y^6-y^8$$

$$0.$$

```
> implicitplot(x^15-21*x^14+8*x^13*y-6*x^13-16*x^12*y+20*x^11*y^2-x^12+
8*x^11*y-36*x^10*y^2+24*x^9*y^3+4*x^9*y^2-16*x^8*y^3+26*x^7*y^4-6*x^6*
y^4+8*x^5*y^5+4*x^3*y^6-y^8=0,x=-10..25,y=-20..100,grid=[200,200]);
```




```
> plot([t^8,t^12+t^14+t^15,t=-1.5..1.3]);
```



C.1.6 Exempel 6

Jämförelse mellan FindImplicitNotation & resultant.

```
> FindImplicitNotation(t^4,t^6+t^7,t,30,true);
```

```
Elapsed Time: 0.000 s.
```

$$-x^7 - 4\,y\,x^5 + x^6 - 2\,y^2\,x^3 + y^4 = 0$$

```
> StartTime:=time();resultant(x-t^4,y-t^6-t^7,t);time()-StartTime;
```

$$StartTime := 24.725$$

$$-y^4 + 2\,x^3\,y^2 - x^6 + 4\,x^5\,y + x^7$$

$$0.$$

Bilaga D

Funktioner för Multiplicitetsföljder

D.1 BlowUpXY

BlowUpXY-funktionen gör en uppblåsning av en funktion $F(x, y)$, och returnerar multipliciteten m av funktionen $F(x, xy)$ och den motsvarande uppblåsningen $F(x, xy)/x^m$.

BlowUpXY(*F*, *XVariable*, *YVariable*)

F	Funktionen som skall blåsas upp.
XVariable	Namnet på variabeln som motsvarar x.
YVariable	Namnet på variabeln som motsvarar y.

$BlowUpXY := \mathbf{proc}(F,\ XVariable,\ YVariable)$

$\quad\quad \mathbf{local}\, G,\ m;$

$\quad\quad G := \mathrm{subs}(YVariable = XVariable * YVariable,\ F)\,;$
$\quad\quad m := \mathrm{ldegree}(G,\ XVariable)\,;$
$\quad\quad \mathrm{RETURN}([m,\ \mathrm{expand}(G/XVariable^m)])$
$\quad \mathbf{end\ proc}$

D.1.1 Exempel 1

```
> BlowUpXY(y^2-x^5,x,y);
```

$$[2,\ y^2 - x^3]$$

D.2 MultiplicitySequenceXY

MultiplicitySequenceXY-funktionen beräknar multiplicitetsföljden av hos en kurva givet på formen $F(x, y) = 0$, där $F \in \mathbb{C}[x, y]$ genom att upprepade gånger anropa *BlowUpXY* tills kruvan blir regulär.

MultiplicitySequenceXY(*F*, *XVariable*, *YVariable*, *PrintSteps*)

F	Funktionen motsvarande multiplicitetsföljden.
XVariable	Namnet på variabeln som motsvarar x.
YVariable	Namnet på variabeln som motsvarar y.
PrintSteps	Frivillig parameter. Om *true* skriver rutinen ut alla steg i beräkningen.

```
MultiplicitySequenceXY := proc(F, XVariable, YVariable)

    local M, m, G, resp;

    if coeff(F, YVariable, 0) = 0 then
      ERROR(`Function not irreducible.`, F)
    end if;

    m := ∞ ;
    M := [] ;
    G := F ;
    while 1 ≤ m do
      resp := BlowUpXY(G, XVariable, YVariable) ;
      m := resp[1] ;
      if G = resp[2] then ERROR(`Infinite loop.`, G) end if;
      G := resp[2] ;
      if 0 < m then M := [op(M), m] end if;
      if 4 ≤ nargs then print([m, G]) end if
    end do;

    RETURN(M)
  end proc
```

D.2.1 Exempel 1

I detta exempel beräknas multiplicitetsföljden för $y^2 - x^5 = 0$:

```
> MultiplicitySequenceXY(y^2-x^5,x,y);
```

$$[2, 2, 1]$$

D.2.2 Exempel 2

I detta exempel beräknas multiplicitetsföljden för $y^3 + x^2y - x^{11} = 0$:

```
> MultiplicitySequenceXY(y^3+x^2*y-x^11,x,y);
```

$$[3, 1, 1, 1, 1, 1, 1, 1, 1]$$

D.2.3 Exempel 3

I detta exempel beräknas multiplicitetsföljden för $y^5x^2+y^7x+y^3x^4-x^{12} = 0$:

```
> MultiplicitySequenceXY(y^5*x^2+y^7*x+y^3*x^4-x^12,x,y);
```

$$[7, 3, 2]$$

D.3 FindIndexes

FindIndexes är en hjälpfunktion till *FindFunctionXYFromMultiplicitySequenece*. Funktionen går rekursivt igenom ett algebraiskt uttryck och tar fram indexen till eventuella konstanter i uttrycket. Exempelvis innehåller uttrycket a_1y indexet 1, och uttrycket $a_2y^4 + a_3y$ indexena 2 och 3.

FindIndexes(*Expression*)

Expression Uttrycket som skall undersökas.

FindIndexes := **proc**(*Expression*)

local a, b, c, n;

a := {op(*Expression*)} ;
b := {} ;
n := nops(a) ;

if $n = 1$ **and** type(a_1, *integer*) **then**
  $b := \{a_1\}$
elif $1 < n$ **then**
  for c **to** n **do**
    if not type(a_c, *integer*) **then**
      $b := b$ union FindIndexes(a_c, *ArrayName*)
    end if
  end do
end if;
RETURN(b)
end proc

D.3.1 Exempel 1

```
> FindIndexes(a[1]*y);
```

$$\{1\}$$

D.3.2 Exempel 2

```
> FindIndexes(a[3]*y^4+y*a[2]);
```

$$\{2, 3\}$$

D.4 FindFunctionXYFromMultiplicitySequence

Denna funktion beräknar en funktion givet på formen $F(x, y) = 0$, där $F(x, y) \in \mathbb{C}[x, y]$ vars multiplicitetsföljd är given i anropet. Funktionen kan också om man vill beräkna en familj med kurvor vars multiplicitetsföljder alla motsvarar den angivna följden. Notera dock att inte alla möjliga sådana funktioner beräknas.

FindFunctionXYFromMultiplicitySequence(*Sequence*, *CalcFamily*, *PrintProgress*)

Sequence	Multiplicitetsföljden.
CalcFamily	*true* eller *false*. Om *true* beräknas en familj med lösningar.
PrintProgress	Frivillig parameter. Om *true* skrivs alla steg ut i beräkningen.

FindFunctionXYFromMultiplicitySequence := **proc**(*Sequence*, *CalcFamily*)

```
local Max, SumS, i, j, k, l, m, n, p, q, f, g, h, a, NrA, Conditions,
    PrintProgress, NonZero, Zero;

PrintProgress := 3 <= nargs and args[3] ;

if not type(Sequence, list) then
    ERROR(`Sequence must be a list of integers.`, Sequence)
end if;

n := nops(Sequence) ;
if n = 0 then
    ERROR(`Empty list!`, Sequence)
end if ;

i := Sequence[1] ;
for j to n do
    if not type(Sequence[j], integer) then
        ERROR(`The list must only contain integer numbers.`, Sequence[j])
    end if;

    if i < Sequence[j] then
        ERROR(`The list must be a descending (not strict) sequence of
                integers.`, Sequence)
    end if;
    i := Sequence[j]
end do;

Max := Sequence[1] ;
SumS := sum(Sequence[v], v = 1..n) ;
NrA := 0 ;
f := -x^SumS ;
for i to Max dofor j from 0 to SumS - i do
        NrA := NrA + 1 ;
        f := a[NrA] * y^i * x^j + f ;
        Zero[NrA] := false ;
        NonZero[NrA] := false
    end do
end do;

if PrintProgress then
    print(f)
end if ;
```

```
g := f ;
for i to n do
    g := subs(y = x * y, g) ;
    if PrintProgress then
        print(g)
    end if;

    k := Sequence_i ;
    for j to k − 1 do
        l := coeff(g, x, j) ;
        if l ≠ 0 then
            if PrintProgress then
                print(l = 0)
            end if;

            l := FindIndexes(l) ;
            m := nops(l) ;
            if 0 < m thenfor p to m do
                    q := l_p ;
                    f := f − coeff(f, a_q, 1) * a_q ;
                    g := g − coeff(g, a_q, 1) * a_q ;
                    Zero_q := true
                end do
            else
                ERROR(‘Invalid function.‘, f)
            end if
        end if
    end do;

    if coeff(g, x, k) = 0 or coeff(g, x, 0) ≠ 0 then
        ERROR(‘Internal error : Unable to find function.
                Starting function too small.‘, f)
    end if;
    g := expand(g/x^k)
end do;

Conditions := [] ;
if PrintProgress then
    print(f)
end if;

g := f ;
h := f ;
```

for i **to** n **do**
  $g := \mathrm{subs}(y = x * y,\ g)$;
  if $\mathit{PrintProgress}$ **then**
    $\mathrm{print}(g)$
  end if;

  $k := \mathit{Sequence}_i$;
  $p := \mathrm{coeff}(g,\ x,\ k)$;
  $l := \mathrm{FindIndexes}(p)$;
  $m := \mathrm{nops}(l)$;
  if $0 < m$ **then**
    for j **to** m **do**
      $q := l_j$;
      if $\mathit{NonZero}_q$ **then**
        $p := 0$
      end if
    end do
  end if;

  if $p \neq 0$ **then**
    $q := \mathrm{degree}(p,\ y)$;
    $j := \mathrm{ldegree}(p,\ y)$;
    if $\mathit{PrintProgress}$ **then**
      $\mathrm{print}([p \neq 0,\ q])$
    end if;

    if $0 < q$ **and** $0 < j$ **then**
      $p := \mathrm{coeff}(p,\ y,\ q)$;
      for j **to** $\mathrm{nops}(\mathit{Conditions})$ **do**
        if $\mathit{Conditions}_j = p$ **then**
          $p := 0$
        end if
      end do;

      if $p \neq 0$ **then**
        $q := \mathrm{coeff}(h,\ p,\ 1)$;
        $h := q + h - q * p$;
        $\mathit{Conditions} := [\mathrm{op}(\mathit{Conditions}),\ p]$;
        $l := \mathrm{FindIndexes}(p)$;
        $m := \mathrm{nops}(l)$;
        if $0 < m$ **then**
          $\mathit{NonZero}_{l_1} := \mathit{true}$
        end if
      end if
    end if
  end if;

```
        g := expand(g/x^k)
    end do;

    l := FindIndexes(h);
    m := nops(l);
    for p to m do
        q := l_p;
        h := h − coeff(h, a_q, 1) ∗ a_q
    end do;

    if CalcFamily then
        for i to nops(Conditions) do
            Conditions_i := Conditions_i ≠ 0
        end do;

        RETURN([h, f, op(Conditions)])
    else
        RETURN(h)
    end if
end proc
```

D.4.1 Exempel 1

Följande exempel beräknar en familj av funktioner $F(x, y) \in \mathbb{C}[x, y]$ sådana att kurvorna $F(x, y) = 0$ har multiplicitetsföljden $\{2, 2, 1\}$.

```
> FindFunctionXYFromMultiplicitySequence([2,2,1],true);
```

$$[y^2 - x^5,\ a_9\, y^2\, x^3 + a_8\, y^2\, x^2 + a_7\, y^2\, x + a_6\, y^2 + a_5\, y\, x^4 + a_4\, y\, x^3 + a_3\, y\, x^2 - x^5,\ a_6 \neq 0]$$

Funktionen visar först en enkel lösning: $y^2 - x^5 = 0$. Notera att detta är samma funktion som i exempel 1 för funktionen *MultiplicitySequenceXY*. En familj med motsvarande multiplicitetsföljd ges också:

$$\begin{cases} a_9\, y^2\, x^3 + a_8\, y^2\, x^2 + a_7\, y^2\, x + a_6\, y^2 + a_5\, y\, x^4 + a_4\, y\, x^3 + a_3\, y\, x^2 - x^5 = 0 \\ a_6 \neq 0 \end{cases}$$

D.4.2 Exempel 2

Följande exempel beräknar en funktion $F(x, y) \in \mathbb{C}[x, y]$ som har multiplicitetsföljden $\{3, 1, 1, 1, 1, 1, 1, 1, 1\}$. Denna följd är tagen från exempel 2 för funktionen *MultiplicitySequenceXY*.

```
> FindFunctionXYFromMultiplicitySequence(
[3,1,1,1,1,1,1,1,1],false);
```

$$y^3 + x^2\,y - x^{11}$$

Som svar fick vi $y^3 + x^2y - x^{11} = 0$, vilket är samma lösning som den given i exempel 2 för funktionen *MultiplicitySequenceXY*.

D.4.3 Exempel 3

Följande exempel beräknar en funktion $F(x,y) \in \mathbb{C}[x,y]$ som har multiplicitetsföljden $\{7,3,2\}$. Denna följd är tagen från exempel 3 för funktionen *MultiplicitySequenceXY*.

```
> FindFunctionXYFromMultiplicitySequence([7,3,2],false);
```

$$x^4\,y^3 - x^{12} + y^7$$

Som svar fick vi $y^7+y^3x^4-x^{12} = 0$. Vi kan testa att multiplicitetsföljden stämmer:

```
> MultiplicitySequenceXY(y^7+y^3x^4-x^12,x,y);
```

$$[7,\ 3,\ 2]$$

D.4.4 Exempel 4

I detta exempel beräknas en kurva för multiplicitetsföljden $\{13, 6, 6, 6, 6, 4\}$:

```
> FindFunctionXYFromMultiplicitySequence([13,6,6,6,6,4],
false);
```

$$y^6\,x^7 + y^{13} - x^{41}$$

Kurvan blir således $y^{13} + y^6x^7 - x^{41} = 0$. Vi testar detta:

```
> MultiplicitySequenceXY(y^13+y^6x^7-x^41,x,y);
```

$$[13,\ 6,\ 6,\ 6,\ 6,\ 4]$$

D.4.5 Exempel 5

Nu betraktar vi en lite mer komplicerad multiplicitetsföljd: $\{4, 3, 2, 1\}$:

```
> FindFunctionXYFromMultiplicitySequence([4,3,2,1],true);
```

$$[y^3\,x - x^{10} + y^4 + x^3\,y^2,$$
$$-x^{10} + a_{34}\,y^4\,x^6 + a_{33}\,y^4\,x^5 + a_{32}\,y^4\,x^4 + a_{31}\,y^4\,x^3 + a_{30}\,y^4\,x^2 +$$
$$a_{29}\,y^4\,x + a_{28}\,y^4 + a_9\,y\,x^8 + a_8\,y\,x^7 + a_7\,y\,x^6 + a_{10}\,y\,x^9 + a_{14}\,y^2\,x^3 +$$
$$a_{15}\,y^2\,x^4 + a_{16}\,y^2\,x^5 + a_{17}\,y^2\,x^6 + a_{18}\,y^2\,x^7 + a_{19}\,y^2\,x^8 + a_{24}\,y^3\,x^4 +$$
$$a_{23}\,y^3\,x^3 + a_{22}\,y^3\,x^2 + a_{21}\,y^3\,x + a_{27}\,y^3\,x^7 + a_{26}\,y^3\,x^6 + a_{25}\,y^3\,x^5,$$
$$a_{28} \neq 0,\ a_{21} \neq 0,\ a_{14} \neq 0]$$

Den enkla lösningen blir således $y^3\,x - x^{10} + y^4 + x^3\,y^2 = 0$. Vi kontrollerar detta:

```
> MultiplicitySequenceXY(y^3*x-x^10+y^4+x^3*y^2,x,y);
```

$$[4,\ 3,\ 2,\ 1]$$

D.4.6 Exempel 6

Svårighetsgraden höjs. Vi betraktar nu multiplicitetsföljden {15, 13, 12, 9, 7, 5, 4, 3, 1}:

```
> FindFunctionXYFromMultiplicitySequence(
[15,13,12,9,7,5,4,3,1],false);
```

$$y^9\,x^{13} + y^3\,x^{44} + y^5\,x^{31} + y^{12}\,x^4 + y^7\,x^{21} + y^{13}\,x^2 + y^4\,x^{37} + y^{15} - x^{69}$$

En lösning är således $y^9\,x^{13} + y^3\,x^{44} + y^5\,x^{31} + y^{12}\,x^4 + y^7\,x^{21} + y^{13}\,x^2 + y^4\,x^{37} + y^{15} - x^{69} = 0$. Vi kontrollerar:

```
> MultiplicitySequenceXY(y^9*x^13+y^3*x^44+y^5*x^31+
y^12*x^4+y^7*x^21+y^13*x^2+y^4*x^37+y^15-x^6,x,y);
```

$$[15,\ 13,\ 12,\ 9,\ 7,\ 5,\ 4,\ 3,\ 1]$$

D.4.7 Exempel 7

Vi betraktar multiplicitetsföljden $\{15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1\}$:

```
> FindFunctionXYFromMultiplicitySequence([15,14,13,12,11,
10,9,8,7,6,5,4,3,2,1],false);
```

$$y^3\,x^{78} + y^6\,x^{45} + y^9\,x^{21} + y^{12}\,x^6 + y^2\,x^{91} + y^{14}\,x + y^7\,x^{36} + y^8\,x^{28} + y^{11}\,x^{10} + y^{13}\,x^3 + y^{10}\,x^{15} + y^5\,x^{55} + y^{15} - x^{120} + y^4\,x^{66}$$

Svaret blir $y^3\,x^{78}+y^6\,x^{45}+y^9\,x^{21}+y^{12}\,x^6+y^2\,x^{91}+y^{14}\,x+y^7\,x^{36}+y^8\,x^{28}+y^{11}\,x^{10} + y^{13}\,x^3 + y^{10}\,x^{15} + y^5\,x^{55} + y^{15} - x^{120} + y^4\,x^{66} = 0$. Vi kontrollerar detta:

```
> MultiplicitySequenceXY(y^3*x^78+y^6*x^45+y^9*x^21+
y^12*x^6+y^2*x^91+y^14*x+y^7*x^36+y^8*x^28+y^11*x^10+
y^13*x^3+y^10*x^15+y^5*x^55+y^15-x^120+y^4*x^66,x,y);
```

$$[15,\ 14,\ 13,\ 12,\ 11,\ 10,\ 9,\ 8,\ 7,\ 6,\ 5,\ 4,\ 3,\ 2,\ 1]$$