Multisectioning, Rational Poly-Exponential Functions and Parallel Computation.

by

Kevin Hare

B.Math, University of Waterloo, 1997.

a thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the Department
of
Mathematics & Statistics.

APPROVAL

Name:	Kevin Hare
Degree:	Master of Science
Title of thesis:	Multisectioning, Rational Poly-Exponential Functions and Parallel Computation.
Examining Committee:	Dr. R. Lockhart Chair
	Dr. J. M. Borwein Senior Supervisor
	Dr. M. Monagan
	Dr. L. Goddyn
	Dr. A. Gupta Department of Computing Science External Examiner
Date Approved:	

Abstract.

Bernoulli numbers and similar arithmetic objects have long been of interest in mathematics. Historically, people have been interested in different recursion formulae that can be derived for the Bernoulli numbers, and the use of these recursion formulae for the calculation of Bernoulli numbers. Some of these methods, which in the past have only been of theoretical interest, are now practical with the availability of high-powered computation.

This thesis explores some of these techniques of deriving new recursion formulae, and expands upon these methods. The main technique that is explored is that of "*multisectioning*". Typically, the calculation of a Bernoulli number requires the calculation of all previous Bernoulli numbers. The method of multisectioning is such that only a fraction of the previous Bernoulli numbers are needed. In exchange, a more complicated recursion formula, called a "*lacunary recursion formula*", must be derived and used.

Dedication.

I would like to dedicate this thesis to my parents, who always supported me with my interest in mathematics.

Acknowledgments.

I would like to thank my supervisor, Jon Borwein, for all his help and insight with respect to this area of research. Also, I would like to thank Marni Mishna, Cindy Loten and Jeff Graham for their proof reading of my thesis, Greg Fee for all of his suggestions on how to improve my Maple code, and numerous other people both within the CECM, and at SFU who made my time here enjoyable.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and preliminaries.

1.1 Introduction.

Bernoulli numbers and similar arithmetic objects have long been of interest in mathematics. Historically, people have been interested in different recursion formulae that can be derived for the Bernoulli numbers, and the use of these recursion formulae for the calculation of Bernoulli numbers. Some of these methods, which in the past have only been of theoretical interest, are now practical with the availability of high-powered computation.

This thesis explores some of these techniques of deriving new recursion formulae, and expands upon these methods. The main technique that is explored is that of "*multisectioning*". Typically, the calculation of a Bernoulli number requires the calculation of all previous Bernoulli numbers. The method of multisectioning is such that only a fraction of the previous Bernoulli numbers are needed. In exchange, a more complicated recursion formula, called a "*lacunary recursion formula*", must be derived and used.

There is a simple formula for $\zeta(n)$, the "*Riemann zeta function*" evaluated at n, for positive even integers n and for negative odd integers n in terms of the Bernoulli numbers. Also, there are numerous constants, (π^{2n}, $\log 2$, γ - the Euler gamma function, τ - the golden mean, G - Catalan's constant) that admit identities of infinite sums of zeta values. Thus the calculations of Bernoulli numbers can be used for certain high precision evaluations of other constants [6].

Bernoulli numbers were first introduced by Jacques Bernoulli (1654-1705), in the second part of his treatise published in 1713, *Ars conjectandi* ("Art of Conjecturing"). At the time, Bernoulli numbers were used for writing the infinite series expansions of hyperbolic and trigonometric functions [7].

Von Staudt and Clausen independently discovered a rapid means of determining the denominator of the Bernoulli numbers [17]. This is very useful for testing to see if the calculation was done without errors. (Any error will most likely return a result for which the Clausen - von Staudt theorem does not hold.)

Van den Berg was the first to discuss finding recurrence formulae for the Bernoulli numbers with arbitrary sized gaps (1881) [19]. (Gaps of size m implies that only $\frac{1}{m}$-th of the information is required, and is the result of multisectioning by m.) Haussner worked on this again, 12 years later (1893) giving the results in terms of hypergeometric functions [19]. Ramanujan, in 1911, is given credit for first giving the formulae for small gaps explicitly. Ramanujan showed how gaps of size 7 could be found, and explicitly wrote out the recursion for gaps of size 6 [4, 19, 22]. These methods were extended to the Euler numbers in 1914 by Glaisher, who used these to compute the first 27 non-zero Euler numbers [14].

Nielsen in 1922, gave an improved notation from a computational point of view to deal with gaps of large sizes [19].

Lehmer in 1934 extended these methods to Euler numbers, Genocchi numbers, and Lucas numbers (1934) [19], and calculated the 196-th Bernoulli number.

The goal in this thesis is to expand these techniques to much more than just Bernoulli and Euler numbers. In general anything that is in the form $\frac{\sum_{i=1}^{n} p_i(x) e^{\lambda_i x}}{\sum_{j=1}^{m} q_j(x) e^{\mu_j x}}$ for polynomials $p_i(x)$, $q_j(x) \in \mathbb{C}[x]$ and constants λ_i, $\mu_j \in \mathbb{C}$ can have the terms of its exponential generating function calculated quickly via multisectioning. This type of function is called a "*rational poly-exponential function*".

This thesis will be looking at examples that are derived from Bernoulli numbers, such as Euler numbers, Genocchi numbers and Lucas numbers. But there are a large variety of other situations where rational poly-exponential functions occur. Some are listed below:

- $(1 + x)(\tan(x) + \sec(x))$ - Boustrophedon transform of sequence 1,1,0,0,0,0,... [21]. Reference number A000756 [25, 26].
- $e^{2x}(\tan(x)+\sec(x))$ - Boustrophedon transform of powers of 2 [21]. Reference number A000752 [25, 26].
- $e^{x}(\tan(x) + \sec(x))$ - Boustrophedon transform of all-1's sequence [21]. Reference number A000667 [25, 26].
- $(1 + x)e^{x}(\tan(x) + \sec(x))$ - Boustrophedon transform of natural numbers [21]. Reference number A000737 [25, 26].
- $\frac{e^{-x}}{(1-x)^3}$ - $a(n) = na(n-1) + (n-2)a(n-2)$ [23]. Reference numbers A000153, M1791, N0706 [25, 26].

- $\frac{e^{-x}}{(1-x)^2}$ - $a(n) = na(n-1) + (n-1)a(n-2)$ [11, 23]. Reference numbers A000255, M2905, N1166 [25, 26].
- $\frac{e^{x}}{(1-x)^2}$ - $\sum_{k=0}^{n}(k+1)!\binom{n}{k}$ [3, 29]. Reference numbers A001339, M2901, N1164 [25, 26].
- $\frac{e^{-x}}{(1-x)^4}$ - $a(n) = na(n-1) + (n-3)a(n-2)$ [23]. Reference numbers A000261, M2949, N1189 [25, 26].
- $\frac{1-e^{x}}{1-2e^{-x}}$ - Simplices in barycentric subdivisions of n-simplex. Reference numbers A002050, M3939, N1622 [25, 26].
- $\frac{1}{2+x-e^{x}}$ - Partition n labeled elements into sets of sizes of at least 2 and order the sets. Reference number A032032 [25, 26].
- The tangent numbers T_n where $\tan z = \sum_{i=0}^{\infty}(-1)^{n+1}\frac{T_{2n+1}z^{2n+1}}{(2n+1)!}$ [5].

These examples, with the exception of the last one, were all found with the help of *The Encyclopedia of Integer Sequences* and its online counterpart [25, 26]. The reference number is the number associated with the sequence within *The Encyclopedia of Integer Sequences.*

Also, although most of the techniques discussed in this thesis are for rational poly-exponential functions in one variable, it is possible to perform multisectioning in a more general setting, such as for the Bernoulli polynomials, or Euler polynomials (the exponential generating function with respect to x of $\frac{xe^{tx}}{e^{x}-1}$ and $\frac{2e^{xt}}{e^{x}+1}$ give the Bernoulli and Euler polynomials respectively as polynomials in t) [2].

The goal of multisectioning by m is to calculate a lacunary recursion formula so that to calculate a term of the exponential generating function of the rational poly-exponential function requires only $\frac{1}{m}$-th of the time and an $\frac{1}{m}$-th of the information when compared with the standard recursion formula. This allows the calculation on m different machines to achieve a theoretical speed up of a factor of m. (In actual fact, experience shows that the speed up will be greater than this, as the reduction in memory requirements will delay thrashing, and the system can better utilize memory management.) Unfortunately for large m it becomes impractical to determine what these lacunary recursion formulae are as the time to determine the recursion formulae and the complexity of these recursion formulae far exceeds the time to calculate these values with smaller gaps.

Hence multisectioning is a method to compute the Bernoulli numbers that does not require any shared memory. This method is limited by the growth in the cost of determining the lacunary recursion formulae. Conversely there are methods which make use of shared memory (or limited message passing) that are not limited by any increase in the complexity of the lacunary recursion formulae. These methods are limited by the effectiveness of the communication between processes. These techniques are called "*recycling methods*" [6].

Included with this thesis are are a description of the computer programs to determine the lacunary recurrence relations for multisectioned poly-exponential functions, programs to determine the lacunary recursion formulae for multisectioned rational poly-exponential function, as well as algorithms to perform these calculations by recycling. For space consideration the actual code was not included within the thesis. These programs can be found on the web at [1]. This is all written in Maple [13].

1.2 Outline.

Chapter 2 defines and explores poly-exponential functions. This chapter examines some closure properties and metrics upon these functions. As well, this chapter looks at some examples of multisectioning functions of this type.

Rational poly-exponential functions are defined and explored in Chapter 3. Again some closure properties, and metrics upon these functions are examined. As well, examples of how to calculate the coefficients of the exponential generating functions of rational poly-exponential functions and multisectioned rational poly-exponential functions via their lacunary recursion formulae are looked at.

Chapter 4 examines different techniques of calculating lacunary recursion formulae for multisectioned poly-exponential functions.

Different techniques of calculating lacunary recursion formulae for multisectioned rational poly-exponential functions are examined at in Chapter 5.

Chapter 6 looks at different methods to perform the calculation of the coefficients of the exponential generating functions of rational poly-exponential functions, after the lacunary recursion formulae are determined. These different techniques take advantage of multi-processor computers, and distributed computer networks.

The last chapter, Chapter 7 discusses some of the results of this thesis, and makes some conclusions as to what has been learned as a result of these investigations.

Appendix A is an outline of the code. Appendix B lists the common notation and page references. Appendix C contains a list of definitions along with the page reference where the definition is first made. Appendix D is for the bugs reports of bugs found in Maple during the course of these investigations. The last appendix, Appendix E is the code.

Chapter 2

Poly-exponential functions.

2.1 Poly-exponential functions.

The study of rational poly-exponential functions is begun with the exploration of a simpler model; that of poly-exponential functions. To that end define:

Definition 2.1 (Poly-exponential function.) *Let λ_1, ..., $\lambda_n \in \mathbb{C}$ be constants and $p_1(x)$, ..., $p_n(x) \in \mathbb{C}[x]$ be polynomials. Then*

$$\sum_{i=1}^{n} p_i(x) e^{\lambda_i x},$$

*is a "*poly-exponential function*". Denote the set of all such functions by $\mathcal{P}$.*

This definition along with Lemma 2.1 and Theorem 2.1 are generalization of examples found in Wilf's *Generating Functionology* [30].

Many results for poly-exponential functions can be extended to ratios of poly-exponential functions, thus allowing a simpler setting for developing techniques for the calculations that are the goal of this thesis. Section 2.2 examines the relationship between exponential generating functions and poly-exponential functions. In Section 2.3 the recurrence polynomial corresponding to a linear recurrence relation is defined and explored. Section 2.4 examines in detail the structure and some of the substructure of $\mathcal{P}$, defining both $\mathcal{P}^{R_1,R_2}$ and $\mathcal{P}_{R_1,R_2}$ ($\mathcal{P}^{R_1,R_2}$ and $\mathcal{P}_{R_1,R_2}$ being subrings of $\mathcal{P}$ where the certain coefficients lie within R_1 or R_2). The relationship between two subrings of $\mathcal{P}$, $\mathcal{P}^{R_1,R_2}$ and $\mathcal{P}_{R_1,R_2}$, and showing that these subrings are distinct are shown in Section 2.5. (The subrings are defined by restricting the coefficients to certain rings.) In Section 2.6 some metrics of complexity are introduced for the functions in $\mathcal{P}$, and the relationships between these metrics, with each other and with standard operations such as addition or multiplication are explored. Section 2.7 contains three detailed examples. The last section, Section 2.8, summarizes the main points of this chapter into a final theorem.

2.2 Exponential generating functions.

The main result of this section is the detailing of the relationship between poly-exponential functions and exponential generating functions.

Lemma 2.1 *Let $s(x)$ be a complex valued function. Then $s(x)$ can be written as an exponential generating function $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$, where the b_i satisfies an N-term linear recurrence relation with constant terms if and only if $s(x)$ can be written as $\sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$ for polynomials $p_i(x) \in \mathbb{C}[x]$ and non-zero constants $\lambda_i \in \mathbb{C}$.*

Proof: Let $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ where the b_i satisfy the linear recurrence relation $b_i = \beta_1 b_{i-1} + ... + \beta_N b_{i-N}$, $\beta_N \neq 0$ for $i \geq N$. Let λ_1, ..., λ_N be roots of the polynomial $x^N - \beta_1 x^{N-1} - ... - \beta_N$ (not necessarily distinct). It is worth noting here that $\lambda_i \neq 0$ for all i. From a standard result on linear recurrence relations [16], it follows that $b_j = \sum_{i=1}^{N} \alpha_i j^{(r_i)} \lambda_i^{j-r_i}$ for some $r_i \in \mathbb{Z}$, and some $\alpha_i \in \mathbb{C}$. Here the notation of Comtet [10] is used, where $j^{(r)} = j(j-1)(j-2)...(j-r+1)$ and $j^{(0)} = 1$. Thus:

$$\begin{aligned} s(x) &= \sum_{j=0}^{\infty} b_j \frac{x^j}{j!} = \sum_{j=0}^{\infty}\sum_{i=1}^{N} \frac{\alpha_i j^{(r_i)} \lambda_i^{j-r_i} x^j}{j!} = \sum_{i=1}^{N}\sum_{j=0}^{\infty} \alpha_i x^{r_i} \left(\frac{j^{(r_i)} \lambda_i^{j-r_i} x^{j-r_i}}{j!}\right) \\ &= \sum_{i=1}^{N} \alpha_i x^{r_i} \sum_{j=0}^{\infty} \left(\frac{j^{(r_i)} \lambda_i^{j-r_i} x^{j-r_i}}{j!}\right) = \sum_{i=1}^{N} \alpha_i x^{r_i} \sum_{j=r_i}^{\infty} \left(\frac{\lambda_i^{j-r_i} x^{j-r_i}}{(j-r_i)!}\right) = \sum_{i=1}^{N} \alpha_i x^{r_i} e^{\lambda_i x}. \end{aligned}$$

Now combine the $\alpha_i x^{r_i}$ which have the same λ_i, and relabel to get $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$, where the λ_i are distinct and non-zero.

To prove the other direction, let $t(x) = \sum_{j=1}^{m} q_j(x)e^{\mu_j x}$, where $\mu_j \neq 0$, $\mu_j \in \mathbb{C}$ and $q_j(x) \in \mathbb{C}[x]$ are polynomials. Consider the polynomial:

$$P(x) = \prod_{j=1}^{n} (x - \mu_j)^{\deg(q_j(x))} = x^n - \alpha_1 x^{n-1} - ... - \alpha_n.$$

Then $t(x) = \sum_{j=0}^{\infty} d_j \frac{x^j}{j!}$ where the d_j satisfies the n term linear recurrence relation $d_j = \alpha_1 d_{j-1} + ... + \alpha_n d_{j-n}$. Later, in Section 2.3 it will be shown that $P(x)$ is the recurrence polynomial of $t(x)$.

■

Theorem 2.1 *Let $s(x)$ be a complex valued function. Then $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ where there exists an m, such that for $i > m$ the b_i satisfy an N-term linear recurrence relation with constant terms if and only if $s(x) \in \mathcal{P}$.*

Proof: First consider $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ where after some m, the b_i satisfy an N-term linear recurrence relation. A degree m polynomial can be extracted, say $p_0(x)$ $(= \sum_{i=0}^{m} \beta_i \frac{x^i}{i!})$ such that the resulting $\bar{b}_i$ $(= b_i - \beta_i)$ satisfy an N-term linear recurrence relation. Then by Lemma 2.1 $s(x)$ can be written as:

$$s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!} = \sum_{i=0}^{\infty} \bar{b}_i \frac{x^i}{i!} + p_0(x) = \sum_{i=1}^{n} p_i(x) e^{\lambda_i x} + p_0(x) e^{0x},$$

for some polynomials $p_i(x)$ and constants λ_i.

Similarly, if $t(x) = \sum_{j=1}^{m} q_j(x) e^{\mu_j x} + p_0(x)$, for polynomials $p_0(x)$, $q_j(x)$, and non-zero constants μ_j, by Lemma 2.1, $t(x)$ can be rewritten as:

$$t(x) = \sum_{j=0}^{\infty} d_j \frac{x^j}{j!} + p_0(x) = \sum_{j=0}^{\infty} \bar{d}_j \frac{x^j}{j!},$$

where the d_j satisfy an N-term linear recurrence relation and where the $\bar{d}_j$ (which are derived by adding the d_j to the coefficients of the polynomial $p_0(x)$) satisfy an N-term linear recurrence relation for $j \geq N + \deg(p_0(x))$.

■

Example 1 *Consider the following example in Maple. For more information about the Maple code, see Appendix A. For the Maple code see Appendix E. The Maple code and help files (including information about syntax) are available on the web at [1].*

```
> with(MS):
```

Consider the function $s_1(x) = x + x\,e^x$. Converting this to an exponential generating function gives:

```
> s[1] := x + x * exp(x):
> convert_egf(s[1], b, x);
```

$$\mathrm{b}(x) = 2\,\mathrm{b}(x-1) - \mathrm{b}(x-2),\ b,\ x,\ [\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 2,\ \mathrm{b}(2) = 2,\ \mathrm{b}(3) = 3]$$

So $s_1(x)$ can be written as $\sum_{i=0}^{\infty} \frac{b_i\,x^i}{i!}$ where $b_i = 2\,b_{i-1} - b_{i-2}$, with $b_0 = 0$, $b_1 = 2$, $b_2 = 2$ and $b_3 = 3$.

Example 2 *Consider the following example in Maple.*

```
>   with(MS):
```

Consider the function $s_2(x) = \sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}$*, where* $b_i = b_{i-1} + b_{i-2}$ *with* $b_0 = 0$ *and* $b_1 = 1$*. These* b_i *are the "*Fibonacci numbers*" [2]. Converting this to a poly-exponential function gives.*

```
>   s[2] := b(x) = b(x-1) + b(x-2), b, x, [b(0) = 0, b(1) = 1];
```

$$s_2 := \mathrm{b}(x) = \mathrm{b}(x-1) + \mathrm{b}(x-2),\ b,\ x,\ [\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 1]$$

```
>   convert_pe(s[2]);
```

$$-\frac{1}{5}\sqrt{5}\, e^{(x\,(1/2 - 1/2\,\sqrt{5}))} + \frac{1}{5}\sqrt{5}\, e^{(x\,(1/2 + 1/2\,\sqrt{5}))},\ x$$

So this can be written as a poly-exponential function, as demonstrated above.

2.3 The recurrence polynomial.

Identifying linear recurrence relations with polynomials will be useful for the further exploration of poly-exponential functions and rational poly-exponential functions. To this end define:

Definition 2.2 (Recurrence polynomial $P^s(x)$.) *Let* $s(x) \in \mathcal{P}$*, where* $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$*, where the* b_i *satisfy an* N*-term linear recurrence relation for all* b_i*,* $i \geq m+N$*, say* $b_i = \alpha_1 b_{i-1} + ... + \alpha_N b_{i-N}$*. For* $m \geq 1$ *assume that for* $i = m + N - 1$*, that* $b_i \neq \alpha_1 b_{i-1} + ... + \alpha_N b_{i-N}$*. Define the "*recurrence polynomial*"* $P^s(x)$ *by:*

$$P^s(x) = x^m(x^N - \alpha_1 x^{N-1} - ... - \alpha_{N-1}x - \alpha_N).$$

Example 3 *Consider the following example in Maple.*

```
>   with(MS):
```

Again consider $s_1(x) = x + e^x\, x$ *from Example 1. This example determines what* $s_1(x)$*'s recurrence polynomial is.*

```
>   s[1] := x + exp(x)*x;
```

$$s_1 := x + x\, e^x$$

```
>   egf := convert_egf(s[1], b, x);
```

$$egf := \mathrm{b}(x) = 2\,\mathrm{b}(x-1) - \mathrm{b}(x-2),\ b,\ x,\ [\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 2,\ \mathrm{b}(2) = 2,\ \mathrm{b}(3) = 3]$$

```
>   convert_poly(egf);
```

$$x^4 - 2x^3 + x^2$$

In contrast consider a random polynomial, and determine what its linear recurrence relation would be.

```
>   poly := randpoly(x);
```

$$poly := -55x^5 - 37x^4 - 35x^3 + 97x^2 + 50x + 79$$

```
>   convert_rec(poly,b,x);
```

$$\mathrm{b}(x) = -\frac{37}{55}\,\mathrm{b}(x-1) - \frac{7}{11}\,\mathrm{b}(x-2) + \frac{97}{55}\,\mathrm{b}(x-3) + \frac{10}{11}\,\mathrm{b}(x-4) + \frac{79}{55}\,\mathrm{b}(x-5)$$

The recurrence polynomial $P^s(x)$ is defined in this way so that it will contain information about when a linear recurrence relation is valid. This construction was suggested by my supervisor, Jon Borwein partly because a useful corollary follows from this definition as a result.

Corollary 1 *If $s(x) \in \mathcal{P}$, $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$, with n distinct λ_i, then:*

$$\deg(P^s(x)) = \sum_{i=1}^{n} (\deg(p_i(x)) + 1).$$

Later, it will be show that this corollary also follows from Lemma 2.5 and is related to the definition of $deg^P(s(x))$ as given in Definition 2.7.

Let $s(x) \in \mathcal{P}$, $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$. It is possible to find more than one linear recurrence relation for the b_i. For example $b_i = b_{i-1} + b_{i-2}$ and $b_i = 2b_{i-2} + b_{i-3}$ are both valid linear recurrence relations for the Fibonacci numbers. Next it is shown how to avoid the ambiguity of which recurrence polynomial or linear recurrence relation to use.

Define the "*length*" of a linear recurrence relation to be the degree of the recurrence polynomial associated with it. (Later it is shown that this is equivalent to the metric deg^P.) Consider the minimal integer $n \geq 0$ such that there is a linear recurrence relation of length n; this gives a unique lower bound to the length of a linear recurrence relation. From this it can be shown that this minimal linear recurrence relation is unique, for if there were two different linear recurrence relations of length N,

$$\begin{aligned} b_i &= \alpha_1 b_{i-1} + \ldots + \alpha_N b_{i-N} \\ \text{and } b_i &= \beta_1 b_{i-1} + \ldots + \beta_N b_{i-N}, \end{aligned}$$

then

$$0 = (\alpha_1 - \beta_1)b_{i-1} + \ldots + (\alpha_N - \beta_N)b_{i-N},$$

which has non-zero terms, hence is a smaller linear recurrence relation, which is a contradiction.

Therefore from the comments above, and the results of Corollary 1, assume that $P^s(x)$ is the unique smallest polynomial associated with the unique linear recurrence relation of minimal length associated with $s(x) \in \mathcal{P}$.

If $P(x)$ and $Q(x)$ are two recurrence polynomials associated with the linear recurrence relation of $s(x) \in \mathcal{P}$ (not necessarily minimal) then $\gcd(P(x), Q(x))$ is also associated with $s(x)$. In fact, any polynomial $P(x)$ such that $P^s(x)|P(x)$ will yield a linear recurrence relation for $s(x)$, albeit not one of minimal length.

2.4 The structure of $\mathcal{P}$.

As yet, $\mathcal{P}$ has only been looked at as a collection of functions. However $\mathcal{P}$ has an internal structure. The main result of this section is to show that $\mathcal{P}$ is a ring. As well, some subrings of $\mathcal{P}$ are examined. Some of the consequences of this are re-examined in Section 4.6 in which calculations over different subrings of $\mathcal{P}$ and $\mathcal{R}$ (to be defined in Chapter 3) are made.

To the best of my knowledge, the subrings of $\mathcal{P}$ in this section have never been examined before, and the results in this section are new.

Definition 2.3 ($\mathcal{P}_{R_1,R_2}$.) *Let R_1 and R_2 be subrings of $\mathbb{C}$. Define*

$$\mathcal{P}_{R_1,R_2} = \{s(x) \in \mathcal{P} : s(x) = \sum_{i=1}^{n} p_i(x) e^{\lambda_i x}, \lambda_i \in R_1, p_i(x) \in R_2[x]\}.$$

Definition 2.4 ($\mathcal{P}^{R_1,R_2}$.) *Let R_1 and R_2 be subrings of $\mathbb{C}$. Define*

$$\mathcal{P}^{R_1,R_2} = \{s(x) \in \mathcal{P} : s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}, P^s(x) \text{ factors in } R_1[x], b_i \in R_2\}.$$

The main result of this section is to show that $\mathcal{P}_{R_1,R_2}$ and $\mathcal{P}^{R_1,R_2}$ are both rings. First some preliminary definitions are made to help discuss multisectioning. The process of multisectioning has had a long history, including Ramanujan, Lehmer, Glaisher [14, 19, 22]. For a more detailed describe of the history, see Section 1.1.

Definition 2.5 *Define $\omega_m = e^{\frac{2\pi i}{m}}$.*

Definition 2.6 (Multisectioning.) *Let $f(x)$ be a function acting on a subset of $\mathbb{C}$. Define $f_m^q(x) = \frac{1}{m} \sum_{i=0}^{m-1} \omega_m^{-iq} f(\omega_m^i x)$.*

The term "*multisectioning*" is used to describe this process [24]. To say a function $s(x)$ is "*multisectioned by m*" means that $s_m^q(x)$ is being discussed for some q. To say a function $s(x)$ is "*multisectioned by m at q*" means that the function $s_m^q(x)$ is being discussed. The term "*lacunary recurrence relation*" is used to describe the linear recurrence relation of a poly-exponential function that has been multisectioned [24].

If $s(x) \in \mathcal{P}$, then it follows that $s_m^q(x) \in \mathcal{P}$. Let $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$, then:

$$\begin{aligned} s_m^q(x) &= \frac{1}{m}\sum_{k=0}^{m-1} \omega_m^{-kq} s(\omega_m^k x) = \frac{1}{m}\sum_{k=0}^{m-1} \omega_m^{-kq} \sum_{i=0}^{\infty} b_i \frac{x^i \omega_m^{ki}}{i!} = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}\frac{1}{m}\sum_{k=0}^{m-1}\omega_m^{-kq}\omega_m^{ki} \\ &= \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}\frac{1}{m}\sum_{k=0}^{m-1}\omega_m^{-kq+ki}. \end{aligned}$$

By noticing that $\frac{1}{m}\sum_{k=0}^{m-1}\omega_m^{-kq+ki}$ is equal to 1 if and only if $q \equiv i \pmod m$ and 0 otherwise, this simplifies to

$$s_m^q(x) = \sum_{i=0}^{\infty} b_{mi+q} \frac{x^{mi+q}}{(mi+q)!}.$$

So the process of multisectioning will isolate certain terms within the power series.

Consider a poly-exponential functions, say $t(x) = \sum_{i=1}^{n} p_i(x) e^{\lambda_i x}$, then a simple calculation shows that $t_m^q(x)$ has the form:

$$t_m^q(x) = \frac{1}{m}\sum_{j=0}^{m-1}\sum_{i=0}^{n} \omega_m^{-jq} p_i(x\omega_m^{-j}) e^{\lambda_i x \omega_m^{-j}}.$$

Rewriting this as $t_m^q(x) = \sum_{j=1}^{\bar{n}} \bar{p}_j(x) e^{\mu_j x}$, shows that, the recurrence polynomial of $t_m^q(x)$ is:

$$P^{t_m^q(x)}(x) = \prod_{j=1}^{\bar{n}} (x - \mu_j)^{\deg(\bar{p}_j(x))}.$$

The set $\{u_j : j = 1...n\}$ is independant of q, (they will run through $\lambda_i \omega_m^j$). By multisectioning, it is possible that the roots will be of a different multiplicity, hence giving a different recurrence polynomial (as shown in the example below). But to do this, a sub-component of the poly-exponential function needs to have a symmetry when shifting by ω_m around the origin, which would result in a different degree for some $\bar{p}_i(x)$. (For example $e^x - e^{-x}$ has a symmetry which shifting by $\omega_2 = -1$ about the origin, as $e^x - e^{-x} = -1(e^{-1x} - e^{-1\times -x})$. For a more detailed discussions of symmetries, see Section 5.4.) For example when multisectioning by two, then the function would need to have an even component or an odd component. The probability of this happening is not very great (measure zero) so, as long as something is known about $s(x)$, then the fact that the recurrence for $s_m^q(x)$ is likely the same as $s_m^{\bar{q}}(x)$ can be taken advantage of; by simplifying the calculation of the lacunary recurrence relation of $s_m^{\bar{q}}(x)$ to checking if the lacunary recurrence relation of $s_m^q(x)$ is valid for the first few initial values.

Example 4 *Consider the following example in Maple.*

```
>   with(MS):
```

This is an example of a poly-exponential function, which when multisectioned by 2 will give a different linear recurrence relation if it is multisectioned at 0 or at 1. Consider the function $\mathrm{s}(x) = e^x + e^{(-x)} + e^{(2\,x)} - e^{(-2\,x)}$.

```
>   s := exp(x)+exp(-x)+exp(2*x)-exp(-2*x);
```

$$\mathrm{s} := e^x + e^{(-x)} + e^{(2\,x)} - e^{(-2\,x)}$$

```
>   `pe/ms`(s, f, x, 2, 0);
```

$$\mathrm{f}(x) = \mathrm{f}(x-2),\ f,\ x,\ [\mathrm{f}(0) = 2,\ \mathrm{f}(1) = 0]$$

```
>   `pe/ms`(s, f, x, 2, 1);
```

$$\mathrm{f}(x) = 4\,\mathrm{f}(x-2),\ f,\ x,\ [\mathrm{f}(0) = 0,\ \mathrm{f}(1) = 4]$$

In the first case the linear recurrence relation is $\mathrm{f}(x) = \mathrm{f}(x-2)$ *and in the second* $\mathrm{f}(x) = 4\,\mathrm{f}(x-2)$.

The notation of Herstein [18] is used with respect to rings and subrings. Let A be a subset of $\mathbb{C}$. Then $\langle A\rangle$ is the smallest subring of $\mathbb{C}$ that contains A. Denote $A^{-1} = \{a^{-1} : a \in A\}$. Let R_1 and R_2 be subrings of $\mathbb{C}$. Denote $R_1R_2 = \{a_1a_2 : a_1 \in R_1 \text{ and } a_2 \in R_2\}$.

Next some closure properties for $\mathcal{P}^{R_1,R_2}$ and $\mathcal{P}_{R_1,R_2}$ are collected.

Lemma 2.2 *Let R_1, R_2, R_3, R_4 and R_5 be subrings of $\mathbb{C}$. If $s(x) \in \mathcal{P}_{R_1,R_2}$, $t(x) \in \mathcal{P}_{R_3,R_4}$ and $\alpha \in R_5$, then:*

1. $s(x)t(x) \in \mathcal{P}_{\langle R1,R3\rangle,R_2R_4}$,
2. $s(x) + t(x) \in \mathcal{P}_{\langle R_1,R_3\rangle,\langle R_2,R_4\rangle}$,
3. $s'(x) \in \mathcal{P}_{R_1,\langle R_1,R_2\rangle}$,
4. $\int_0^x s(y)dy \in \mathcal{P}_{R_1,\langle \mathbb{Q}R_2,R_1^{-1}R_2\rangle}$,
5. $s(\alpha x) \in \mathcal{P}_{R_1R_5,\langle R_2,R_2R_5\rangle}$,
6. $s_m^q(x) \in \mathcal{P}_{\langle\omega_m\rangle R_1,\langle\frac{1}{m}\rangle\langle\omega_m\rangle R_2}$.

Proof: Assume that $s(x) = \sum_{i=1}^n p_i(x)e^{\lambda_i x}$, and $t(x) = \sum_{j=1}^m q_j(x)e^{\mu_j x}$ throughout this proof.

1. Observe that:
$$s(x)t(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}\sum_{j=1}^{m} q_j(x)e^{\mu_j x} = \sum_{i=1,j=1}^{i=n,j=m} p_i(x)q_j(x)e^{(\lambda_i+\mu_j)x}.$$
Then $p_i(x)q_j(x) \in R_2R_4[x]$, and $\lambda_i + \mu_j \in \langle R_1, R_3\rangle$. So $s(x)t(x) \in \mathcal{P}_{\langle R_1,R_3\rangle,R_2R_4}$.

2. Observe that:
$$s(x) + t(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x} + \sum_{j=1}^{m} q_j(x)e^{\mu_j x}.$$
Both $p_i(x)$ and $q_j(x)$ are in $\langle R_2, R_4\rangle[x]$ and further $\lambda_i, \mu_j \in \langle R_1, R_3\rangle$. Thus $s(x) + t(x) \in \mathcal{P}_{\langle R_1,R_3\rangle,\langle R_2,R_4\rangle}$.

3. Observe that:
$$s'(x) = \sum_{i=1}^{n} \lambda_i p_i(x)e^{\lambda_i x} + \sum_{i=1}^{n} p_i'(x)e^{\lambda_i x}.$$
Consequently $p_i'(x), \lambda_i p_i(x) \in \langle R_2, R_1R_2\rangle[x]$ and that $\lambda_i \in R_1$. Thus $s'(x) \in \mathcal{P}_{R_1,\langle R_2,R_1R_2\rangle}$.

4. Re-index the function $s(x)$ so that $s(x) = \sum_{i=1,\lambda_i\neq 0}^{n} \alpha_i x^{r_i} e^{\lambda_i x} + \sum_{i=0}^{m} \beta_i x^i$, where $\lambda_i \in R_1$, α_i, $\beta_i \in R_2$ and $r_i \in \mathbb{Z}$, $r_i \geq 0$. Then:
$$\begin{aligned}\int_0^x s(y)dy &= \int_0^x \sum_{i=1,\lambda_i\neq 0}^{n} \alpha_i y^{r_i} e^{\lambda_i y} + \sum_{i=0}^{m} \beta_i y^i dy \\ &= \sum_{i=1,\lambda_i\neq 0}^{n} \int_0^x \alpha_i y^{r_i} e^{\lambda_i y} dy + \sum_{i=0}^{m} \int_0^x \beta_i y^i dy \\ &= \sum_{i=1,\lambda_i\neq 0}^{n} \alpha_i \sum_{j=0}^{r_i} \frac{r_i! x^{r_i-j} e^{\lambda_i x}(-1)^j}{\lambda_i^{j+1}(j-1)!} + \sum_{i=0}^{m} \frac{\beta_i x^{i+1}}{i+1} \\ &= \sum_{i=1,\lambda_i\neq 0}^{n} \alpha_i e^{\lambda_i x} \sum_{j=0}^{r_i} \frac{r_i! x^{r_i-j}(-1)^j}{\lambda_i^{j+1}(j-1)!} + \sum_{i=0}^{m} \frac{\beta_i x^{i+1}}{i+1}.\end{aligned}$$

The case $\lambda_i \neq 0$ gives that the coefficients are contained in the subring $\langle R_2R_1^{-1}\rangle$. In the case $\lambda_i = 0$, the coefficients are contained in $\mathbb{Q}R_2$. The λ_i are still in R_1. Therefore $\int_0^x s(y)dy \in \mathcal{P}_{R_1,\langle \mathbb{Q}R_2, R_2R_1^{-1}\rangle}$.

5. Notice that $s(\alpha x) = \sum_{i=1}^{n} p_i(\alpha x)e^{\alpha\lambda_i x}$. So $p_i(\alpha x) \in \langle R_2, R_2R_5\rangle$. Further $\alpha\lambda_i \in R_1R_5$, so $s(\alpha x) \in \mathcal{P}_{R_1R_5,\langle R_2,R_2R_5\rangle}$.

6. By combining part 2 and part 5 of this lemma $s_m^q(x)$ can be written as:
$$\frac{1}{m}\sum_{i=1}^{m} \omega_m^{-iq} s(\omega_m^i x) \in \mathcal{P}_{\langle\langle\omega_m\rangle R_1,\langle\omega_m^2\rangle R_1,\ldots\langle\omega_m^m\rangle R_1\rangle,\langle\langle\frac{1}{m}\omega_m\rangle R_2,\langle\frac{1}{m}\omega_m^2\rangle R_2,\ldots\langle\frac{1}{m}\omega_m^m\rangle R_2\rangle}.$$
This simplifies to $\mathcal{P}_{\langle\omega_m\rangle R_1,\langle\frac{1}{m}\rangle\langle\omega_m\rangle R_2}$.

■

Lemma 2.3 *Let R_1, R_2, R_3, R_4 and R_5 be subrings of $\mathbb{C}$. If $s(x) \in \mathcal{P}^{R_1,R_2}$, $t(x) \in \mathcal{P}^{R_3,R_4}$, and $\alpha \in R_5$ then:*

1. $s(x)t(x) \in \mathcal{P}^{\langle R_1,R_3\rangle,R_2R_4}$,
2. $s(x)+t(x) \in \mathcal{P}^{\langle R_1,R_3\rangle,\langle R_2,R_4\rangle}$,
3. $s'(x) \in \mathcal{P}^{R_1,R_2}$,
4. $\int_0^x s(y)dy \in \mathcal{P}^{R_1,R_2}$,
5. $s(\alpha x) \in \mathcal{P}^{R_1R3,\langle R_2,R_2R_3\rangle}$,
6. $s_m^q(x) \in \mathcal{P}^{R_1\langle\omega_m\rangle,R_2}$.

Proof: Again, assume that $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!} = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$, and $t(x) = \sum_{j=0}^{\infty} d_j \frac{x^j}{j!} = \sum_{j=1}^{m} q_j(x)e^{\mu_j x}$ throughout this proof.

1. Consider:

$$s(x)t(x) \quad = \quad \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}\sum_{j=1}^{m} q_j(x)e^{\mu_j x} = \sum_{i=1 j=1}^{i=n,j=m} p_i(x)q_j(x)e^{(\lambda_i+\mu_j)x}.$$

From this $\prod_{i=1,j=1}^{i=n,j=m}(x-\lambda_i-\mu_j)^{\deg(p_i(x))+\deg(q_i(x))}$ is a recurrence polynomial (not necessarily minimal) for $s(x)t(x)$. Hence:

$$P^{st}(x)|\prod_{i=1,j=1}^{i=n,j=m}(x-\lambda_i-\mu_j)^{\deg(p_i(x))+\deg(q_i(x))}.$$

This splits in $\langle R_1, R_3\rangle$. Further,

$$s(x)t(x) \quad = \quad \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}\sum_{j=0}^{\infty} d_j \frac{x^j}{j!} = \sum_{j=0}^{\infty}\sum_{i=0}^{j}\frac{b_{j-i}d_i}{(j-i)!i!}x^j = \sum_{j=0}^{\infty}\sum_{i=0}^{j} b_{j-i}d_i\binom{j}{i}\frac{x^j}{j!}.$$

Therefore the coefficients are in R_2R_4. Thus $s(x)t(x) \in \mathcal{P}^{\langle R_1,R_3\rangle,R_2R_4}$.

2. Consider:

$$s(x)+t(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x} + \sum_{j=1}^{m} q_j(x)e^{\mu_j x}.$$

The polynomial $\prod_{i=1}^{n}(x-\lambda_i)^{\deg(p_i(x))}\prod_{j=1}^{m}(x-\mu)^{\deg(q_i(x))}$ is a recurrence polynomial for $s(x)+t(x)$ (not necessarily minimal). Hence $P^{s+t}(x)|P^s(x)P^t(x)$. Thus $P^{s+t}(x)$ will split in $\langle R_1, R_3\rangle$. Further,

$$s(x)+t(x) \quad = \quad \sum_{i=0}^{\infty} b_i \frac{x^i}{i!} + \sum_{j=0}^{\infty} d_j \frac{x^j}{j!} = \sum_{i=0}^{\infty}(b_i+d_i)\frac{x^i}{i!}.$$

Where the $b_i + d_i$ are in $\langle R_2, R_4\rangle$. Hence $s(x)+t(x) \in \mathcal{P}^{\langle R_1,R_3\rangle,\langle R_2,R_4\rangle}$.

3. If $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$. then

$$s'(x) = \sum_{i=1}^{\infty} b_i \frac{x_{i-1}}{(i-1)!} = \sum_{i=0}^{\infty} b_{i+1}\frac{x_i}{i!}.$$

Hence the coefficients are in the same ring as before, hence in R_2.

Now consider $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$. This implies that:

$$s'(x) = \sum_{i=1}^{n} q_i(x)e^{\lambda_i x},$$

where $\deg(p_i(x)) = \deg(q_i(x))$ if $\lambda_i \neq 0$ and $\deg(p_i(x)) = \deg(q_i(x)) + 1$ if $\lambda_i = 0$. Thus $P^{s'}(x) = \prod_{i=1}^{n}(x-\lambda_i)^{\deg(q_i(x))}$. Therefore if there exists a λ_i that is equal to 0, then $P^{s'}(x) = P^s(x)x$, and otherwise $P^{s'}(x) = P^s(x)$. So $P^{s'}(x)$ splits over the same field as $P^s(x)$. Hence $s'(x) \in \mathcal{P}^{R_1,R_2}$.

4. By observing that

$$\int_0^x s(y)dy \quad = \quad \int_0^x \sum_{i=0}^{\infty} b_i \frac{x^i}{i!} = \sum_{i=0}^{\infty}\int_0^x \frac{b_i}{i!} y^i dy = \sum_{i=0}^{\infty} \frac{b_i}{(i+1)!} y^{i+1}|_0^x = \sum_{i=1}^{\infty}\frac{b_{i-1}}{i!}x^i,$$

it follows that all the coefficients of $\int_0^x s(y)dy$ are in R_2

Now consider $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$. This implies that:

$$\int_0^x s(y)dy = \sum_{i=1}^{n} q_i(x)e^{\lambda_i x},$$

where $\deg(p_i(x)) = \deg(q_i(x))$ if $\lambda_i \neq 0$ and $\deg(p_i(x)) + 1 = \deg(q_i(x))$ if $\lambda_i = 0$. From this $P^{s'}(x) = \prod_{i=1}^{n}(x-\lambda_i)^{\deg(q_i(x))}$. Consequently if there exists a λ_i that is equal to 0, then $P^{\int_0^x s(y)dy}(x) = P^s(x)$, and otherwise $P^{\int_0^x s(y)dy}(x) = P^s(x)$. So $P^{\int_0^x s(y)dy}(x)$ splits over the same field as $P^s(x)$. Thus $\int_0^x s(y)dy \in \mathcal{P}^{R_1,R_2}$.

5. It can be seen that

$$s(\alpha x) = \sum_{i=0}^{\infty} b_i \frac{\alpha^i x^i}{i!},$$

Consequently all of the $b_i\alpha^i \in \langle R_2, R_2R_5\rangle$.

The next aim is to find a linear recurrence relation for the $b_i\alpha^i$. Now if:

$$P^s(x) = x^n - \beta_1 x^{n-1} - ... - \beta_n = \prod_{i=1}^{n}(x-\lambda_i)^{\deg(p_i(x))},$$

and letting $c_i = b_i\alpha^i$ then:

$$\frac{c_i}{\alpha^i} = \beta_1\frac{c_{i-1}}{\alpha^{i-1}} + ... + \beta_n\frac{c_{i-n}}{\alpha^{m-i}}.$$

Multiplying through by α^i gives:

$$c_i = \alpha\beta_1 c_{i-1} + ... + \alpha^n \beta_n c_{i-n},$$

which gives:

$$P^{s(\alpha x)}(x) = x^n - \beta_1 \alpha x^{n-1} - ...\alpha^n \beta_n,$$

this factors as:

$$P^{s(\alpha x)}(x) = \prod_{i=1}^{n} (x - \lambda_i \alpha)^{\deg(p_i(x))}.$$

Therefore $P^{s(\alpha x)}(x)$ splits over $R_1 R_5$. So $s(\alpha x) \in \mathcal{P}^{R_1 R_5, \langle R_2, R_2 R_5 \rangle}$.

6. By combining part 2 and part 5 of this lemma $s_m^q(x)$ can be written as:

$$\frac{1}{m} \sum_{i=1}^{m} \omega_m^{-i*q} s(\omega_m^i x) \in \mathcal{P}^{\langle \langle \omega_m \rangle R_1, \langle \omega_m^2 \rangle R_1, ... \langle \omega_m^m \rangle R_1 \rangle, \langle \frac{1}{m} \langle \omega_m \rangle R_2, \langle \frac{1}{m} \omega_m^2 \rangle R_2, ... \langle \frac{1}{m} \omega_m^m \rangle R_2 \rangle}.$$

But this will simplify to $\mathcal{P}^{\langle \omega_m \rangle R_1, \langle \frac{1}{m} \rangle \langle \omega_m \rangle R_2}$.

An even tighter bound on the coefficients can be seen by noticing that:

$$s_m^q(x) = \sum_{i=0}^{\infty} b_{mi+q} \frac{x^{mi+q}}{(mi+q)!}.$$

From this all the coefficients in the resulting formula are still contained within the ring R_2. Hence $s_m^q(x) \in \mathcal{P}^{R_1 \langle \omega_m \rangle, R_2}$.

■

In the proof of Lemma 2.3, some intermediate results were obtained, which are summarized below:

Corollary 2 *Let R_1, R_2 and R_3 be subrings of $\mathbb{C}$. If $s(x)$, $t(x) \in \mathcal{P}$ such that $P^s(x) \in R_1[x]$, $P^t(x) \in R_2[x]$ and $\alpha \in R_3$. Then:*

1. *$P^{st}(x) \in R_1 R_2[x]$,*

2. *$P^{s+t}(x) \in R_1 R_2[x]$,*

3. *$P^{s'}(x) \in R_1[x]$ (infact $P^s(x) = P^{s'}(x)$ or $P^s(x) = xP^{s'}(x)$),*

4. *$P^{\int_0^x s(y)dy}(x) \in R_1[x]$ (infact $P^{\int_0^x s(y)dy}(x) = P^s(x)$ or $P^{\int_0^x s(y)dy}(x) = xP^s(x)$),*

5. *$P^{s(\alpha x)}(x) \in \langle R_1, R_3 \rangle[x]$,*

6. *$P^{s_m^q}(x) \in R_1[x]$.*

These results will be useful later in Chapters 4 and 5. These results imply that the calculations can normally be assumed to be over "nice" rings such as the integers or rationals.

Corollary 3 *Let R_1 and R_2 be subrings of $\mathbb{C}$. Then $\mathcal{P}_{R_1,R_2}$ and $\mathcal{P}^{R_1,R_2}$ are both rings.*

Example 5 *Consider the following example in Maple.*

```
> with(MS):
```

Consider the function $\mathrm{s}(x) = \sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}$*, where* $b_i = b_{i-1} + b_{i-2}$ *and* $b_0 = 2$, $b_1 = 1$. *These are the Lucas numbers as defined by Graham, Knuth and Patashnik, [16, 24]. To avoid confusion with the Lucas numbers as defined by Lehmer, call these the "*Lucas numbers, type I*" . Now multisection s(x) by 4 at 1.*

```
> s := b(x) = b(x-1) + b(x-2), b, x, [b(0) = 2, b(1) = 1];
```

$$\mathrm{s} := \mathrm{b}(x) = \mathrm{b}(x-1) + \mathrm{b}(x-2),\, b,\, x,\, [\mathrm{b}(0) = 2,\, \mathrm{b}(1) = 1]$$

First convert this to poly-exponential form:

```
> pe := convert_pe(s)[1];
```

$$pe := e^{(x\,(1/2-1/2\,\sqrt{5}))} + e^{(x\,(1/2+1/2\,\sqrt{5}))}$$

Now multisection the poly-exponential function using the formula as given in Definition 2.6.

```
> ms := 1/4*sum(subs(x=x*exp(2*Pi*I/4*i), pe)*exp(-2*Pi*I/4*i), i=0..3);
```

$$\begin{aligned} ms := {} & \frac{1}{4}\,e^{(x\,\%2)} + \frac{1}{4}\,e^{(x\,\%1)} - \frac{1}{4}\,I\,(e^{(I\,x\,\%2)} + e^{(I\,x\,\%1)}) - \frac{1}{4}\,e^{(-x\,\%2)} - \frac{1}{4}\,e^{(-x\,\%1)} \\ & + \frac{1}{4}\,I\,(e^{(-I\,x\,\%2)} + e^{(-I\,x\,\%1)}) \\ & \%1 := \frac{1}{2} + \frac{1}{2}\,\sqrt{5} \\ & \%2 := \frac{1}{2} - \frac{1}{2}\,\sqrt{5} \end{aligned}$$

Now convert this back into an exponential generating function.

```
> convert_egf(ms, b, x);
```

$$\begin{aligned} & \mathrm{b}(x) = -\mathrm{b}(x-8) + 7\,\mathrm{b}(x-4),\, b,\, x, \\ & \quad [\mathrm{b}(0) = 0,\, \mathrm{b}(1) = 1,\, \mathrm{b}(2) = 0,\, \mathrm{b}(3) = 0,\, \mathrm{b}(4) = 0,\, \mathrm{b}(5) = 11,\, \mathrm{b}(6) = 0,\, \mathrm{b}(7) = 0] \end{aligned}$$

From this it follows that $s_4^1(x) = \sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}$*, where* $b_i = 7\, b_{i-4} - b_{i-8}$ *and* $b_1 = 1$, $b_5 = 11$ *and* $b_i = 0$ *if* $i \neq 1 \bmod 4$*. So* $s_4^1(x) = \sum_{i=0}^{\infty} \frac{b_{4\,i+1}\, x^{(4\,i+1)}}{(4\,i+1)!}$.

Alternatively there is automated code to achieve the same result, using this naive method.

```
>   `egf/ms/naive`(s,4,1);
```

$$\mathrm{b}(x) = -\mathrm{b}(x-8) + 7\,\mathrm{b}(x-4),\ b,\ x,$$
$$[\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 1,\ \mathrm{b}(2) = 0,\ \mathrm{b}(3) = 0,\ \mathrm{b}(4) = 0,\ \mathrm{b}(5) = 11,\ \mathrm{b}(6) = 0,\ \mathrm{b}(7) = 0]$$

This is a relationship for the Lucas numbers, type I that is only concerned with $b_1, b_5, b_9, ...$

Automating the process of multisectioning is covered in Chapter 4.

2.5 Hierarchy of $\mathcal{P}$.

While many results for both $\mathcal{P}^{R_1,R_2}$ and $\mathcal{P}_{R_1,R_2}$ have been obtained, it is not yet clear as to how these two rings relate to each other. This section shows that they are in fact different sets of rings. Further an inclusion relationship between the rings is shown.

Theorem 2.2 *Let* R_1 *and* R_2 *be subrings of* $\mathbb{C}$*. Then the following inclusion relationships of the subrings of* $\mathcal{P}$ *hold.*

1. $\mathcal{P}_{R_1,R_2} \subseteq \mathcal{P}^{R_1,\langle R_1 R_2, R_2\rangle}$,
2. $\mathcal{P}^{R_1,R_2} \subseteq \mathcal{P}_{R_1,R_2\langle R_1^{-1},R_1\rangle}$.

Proof:

1. Let $s(x) \in \mathcal{P}_{R_1,R_2}$, $s(x) = \sum_{i=1}^{n} p_i(x) e^{\lambda_i x}$, $p_i(x) \in R_2[x]$, $\lambda_i \in R_1$. Noticing that $P^s(x) = \prod_i^n (x-\lambda_i)^{\deg(p_i(x))}$, demonstrates that $P^s(x)$ splits in $R_1[x]$. Now notice that $b_i = s^{(i)}(0)$, the i-th derivative of $s(x)$. But $s^{(i)}(x) \in \mathcal{P}_{R_1,\langle R_1 R_2, R_2\rangle}$ by Lemma 2.2 part 3. Evaluating at 0 gives $b_i \in \langle R_1 R_2, R_2\rangle$, hence $\mathcal{P}_{R_1,R_2} \subseteq \mathcal{P}^{R_1,\langle R_1 R_2, R_2\rangle}$.

2. Let $s(x) \in \mathcal{P}^{R_1,R_2}$, $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$. By definition $P^s(x)$ splits in R_1. Lemma 2.1 implies that if $s(x) = \sum_i^n \alpha_i x^{(r_i)} e^{\lambda_i x}$ then all the λ_i are in R_1.

   Again from Lemma 2.1 it follows that:

$$b_j = \sum_{i=1}^{n} j^{(r_i)} \lambda_i^{j-r_i} \alpha_{r_i},$$

where $\lambda_i \in R_1$, $j^{(r_i)} \in \mathbb{Z}$ and $b_j \in R_2$, and $j^{(r)} = (j)(j-1)...(j-r+1)$. A solution to these equations using Gaussian elimination requires only addition, subtraction, multiplication, and division of elements in R_1. Thus $\alpha_{r_i} \in R_2\langle R_1^{-1}, R_1\rangle$. Hence $\mathcal{P}^{R_1,R_2} \subseteq \mathcal{P}_{R_1,R_2\langle R_1^{-1},R_1\rangle}$.

■

Consider the following examples, which shows that the two rings are distinct.

Example 6 *Consider $s(x) = e^{\sqrt{2}x} \in \mathcal{P}_{\mathbb{Q}(\sqrt{2}),\mathbb{Z}}$. Now $s'(x) = \sqrt{2}e^{\sqrt{2}x}$, and $\sqrt{2} \notin \mathbb{Z}$ implies that $s'(x) \notin \mathcal{P}_{\mathbb{Q}(\sqrt{2}),\mathbb{Z}}$. But all rings of the form $\mathcal{P}^{R_1,R_2}$ are closed under differentiation (Lemma 2.3). Hence there do not exist rings R_1, R_2 such that $\mathcal{P}_{\mathbb{Q}(\sqrt{2}),\mathbb{Z}} = \mathcal{P}^{R_1,R_2}$.*

Example 7 *Consider $\mathcal{P}^{\mathbb{Z},\mathbb{Z}}$. The goal here is to show that there do not exist rings R_1, R_2 such that $\mathcal{P}^{\mathbb{Z},\mathbb{Z}} = \mathcal{P}_{R_1,R_2}$. Consider the exponential generating function $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ with the linear relation $b_i = 3cb_{i-1} - 2c^2 b_{i-2}$, for $c \in \mathbb{Z}$. If b_0, $b_1 \in \mathbb{Z}$, then $s(x) \in \mathcal{P}^{\mathbb{Z},\mathbb{Z}}$. But this is equivalent to $s(x) = \alpha_1 e^{cx} + \alpha_2 e^{2cx}$, where $\alpha_1 = 2b_0 - \frac{b_1}{c}$ and $\alpha_2 = -b_0 + \frac{b_1}{c}$. Hence α_1 can be any arbitrary rational in $\mathbb{Q}$, say $\frac{p}{q}$, by picking $b_0 = 0$, $b_1 = -p$ and $c = q$. Thus if $\mathcal{P}^{\mathbb{Z},\mathbb{Z}} \subseteq \mathcal{P}_{R_1,R_2}$, then $\mathbb{Z} \subseteq R_1$ (as R_1 must contain arbitrary c, where $c \in \mathbb{Z}$) and $\mathbb{Q} \subseteq R_2$. Now $\mathcal{P}^{\mathbb{Z},\mathbb{Z}}$ is a strict subset of $\mathcal{P}_{\mathbb{Z},\mathbb{Q}}$, as $\frac{1}{2} \in \mathcal{P}_{\mathbb{Z},\mathbb{Q}}$ and $\frac{1}{2} \notin \mathcal{P}^{\mathbb{Z},\mathbb{Z}}$. Hence there do not exist rings R_1 and R_2, such that $\mathcal{P}^{\mathbb{Z},\mathbb{Z}} = \mathcal{P}_{R_1,R_2}$.*

Corollary 4 *If F_1 is a subfield of $\mathbb{C}$, and $R_1 \subseteq F_1$ is a subring of $\mathbb{C}$ then $\mathcal{P}_{R_1,F_1} = \mathcal{P}^{R_1,F_1}$.*

2.6 Some complexity bounds.

Understanding the complexity of the functions being manipulated is useful for doing computations on $s(x) \in \mathcal{P}$. To this end some metrics of complexity are defined. These metrics have been looked at in the past but not in such a generalized fashion. Typically they would be applied to a particular problem, such as the Bernoulli numbers [9].

Definition 2.7 *Let $s(x) \in \mathcal{P}$, where $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$. Define the following metrics:*

1. *$deg^d(s(x)) = \max(\deg(p_i(x)))$,*
2. *$deg^P(s(x)) = \deg(P^s(x))$.*

Example 8 *Consider the following example in Maple.*

```
>  with(MS):
```

This example determines what $deg^d(s(x))$ and $deg^P(s(x))$ are for various s(x). This example uses the automated code described in appendix A.

First consider the function from Example 1.

```
> s[1] := x + x * exp(x);
```

$$s_1 := x + x\,e^x$$

```
> `pe/metric/d`(s[1],x);
```

$$1$$

Recalls that $P^{s_1}(x) = x^4 - 2x^3 + x^2$.

```
> `pe/metric/P`(s[1],x);
```

$$4$$

Next, consider the Fibonacci numbers from Example 2.

```
> s[2] := b(x) = b(x-1)+b(x-2),b,x, [b(0)=0,b(1)=1];
```

$$s_2 := \mathrm{b}(x) = \mathrm{b}(x-1) + \mathrm{b}(x-2),\, b,\, x,\, [\mathrm{b}(0) = 0,\, \mathrm{b}(1) = 1]$$

```
> `egf/metric/d`(s[2]);
```

$$0$$

```
> `egf/metric/P`(s[2]);
```

$$2$$

These metrics are of use later on in Chapter 4 and 5. In those two chapters, upper bounds for functions under different operations are required.

Lemma 2.4 *Let $s(x)$, $t(x) \in \mathcal{P}$, and $\alpha \neq 0$ a constant. Then:*

1. $deg^d(s(x)t(x)) = deg^d(s(x)) + deg^d(t(x))$,
2. $0 \leq deg^d(s(x) + t(x)) \leq \max(deg^d(s(x)), deg^d(t(x)))$,
3. $deg^d(s(x)) - 1 \leq deg^d(s'(x)) \leq deg^d(s(x))$,
4. $deg^d(s(x)) \leq deg^d(\int_0^x s(y)dy) = deg^d(s(x)) + 1$,
5. $deg^d(s(\alpha x)) = deg^d(s(x))$,
6. $0 \leq deg^d(s^q_m(x)) \leq deg^d(s(x))$.

Proof: Write $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$ and $t(x) = \sum_{j=1}^{m} q_j(x)e^{\mu_j x}$ for the remainder of this proof.

1. Notice that:
$$deg^d(s(x)t(x)) = deg^d(\sum_{i=1,j=1}^{i=n,j=m} p_i(x)q_j(x)e^{(\lambda_i+\mu_j)x}).$$
Denote $I = \{i : \deg(p_i(x)) = deg^d(s(x))\}$ and $J = \{j : \deg(q_i(x)) = deg^d(t(x))\}$. Pick $\lambda = \max_{i\in I}(\lambda_i)$ and $\mu = \max_{j\in J}(\lambda_i)$. (The maximum is taken lexigraphically, for example, for two complex numbers α and β, α is greater than β if the real component of α is greater than that of β, or if the real component of α and β are equal, and the imaginary component of α is greater than that of β.)
Consequently the polynomial associated with $\lambda + \mu$ is of degree $deg^d(s(x)) + deg^d(t(x))$. Thus $deg^d(s(x)t(x)) = deg^d(s(x)) + deg^d(t(x))$.

2. The upper bound is clear, and taking $s(x) = -t(x)$ gives the lower bound.

3. Notice that:
$$deg^d(s'(x)) = deg^d(\frac{d}{dx}\sum_{i=1}^{n} p_i(x)e^{\lambda_i x}) = deg^d(\sum_{i=1}^{n}(\lambda_i p_i(x) + p_i'(x))e^{\lambda_i x}).$$
Notice that $\deg(p_i(x)\lambda_i + p_i'(x)) = \deg(p_i(x))$ if $\lambda_i \neq 0$, and is equal to $\deg(p_i(x)) - 1$ if $\lambda_i = 0$. Hence $deg^d(s'(x)) = deg^d(s(x))$ or $deg^d(s(x)) - 1$.

4. Part 4 of Lemma 2.2 shows that:
$$deg^d(\int_0^x s(y)dy) = deg^d(\int_0^x \sum_{i=1}^{n} p_i(y)e^{\lambda_i y}dy) = deg^d(\sum_{i=1}^{n} q_i(x)e^{\lambda_i x}).$$
Where $\deg(q_i(x)) = \deg(p_i(x))$ if $\lambda_i \neq 0$ and $\deg(q_i(x)) = \deg(p_i(x)) + 1$ if $\lambda_i = 0$. Thus $deg^d(\int_0^x s(y)dy) = deg^d(s(x))$ or $deg^d(s(x)) + 1$.

5. Observe that:
$$deg^d(s(\alpha x)) = deg^d(\sum_{i=1}^{n} p_i(\alpha x)e^{\lambda_i \alpha x}).$$
As $\deg(p_i(\alpha x)) = \deg(p_i)$ it follows that $deg^d(s(\alpha x)) = deg^d(s)$.

6. Part 2 and part 5 of this lemma, in combination shows that $deg^d(s_m^q(x)) \leq deg^d(s(x))$. If $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ and $b_i = 0$ whenever $i \equiv q \pmod{m}$, then $s_m^q(x) = 0$. Hence $deg^d(s_m^q(x)) = 0$ in this case.

■

Lemma 2.5 *Let $s(x), t(x) \in \mathcal{P}$, and α a constant. Then:*

1. $deg^P(s(x)t(x)) \leq deg^P(s(x))deg^P(t(x))$,

2. $0 \leq deg^P(s(x)+t(x)) \leq deg^P(s(x)) + deg^P(t(x))$,

3. $deg^P(s(x)) - 1 \leq deg^P(s'(x)) = deg^P(s(x))$,

4. $deg^P(s(x)) \leq deg^P(\int_0^x s(y)dy) = deg^P(s(x)) + 1$,

5. $deg^P(s(\alpha x)) = deg^P(s(x))$,

6. $0 \leq deg^P(s_m^q(x)) \leq m \times deg^P(s(x))$.

Proof: Write $s(x) = \sum_{i=1}^n p_i(x)e^{\lambda_i x}$ and $t(x) = \sum_{j=1}^m q_j(x)e^{\mu_j x}$ for the remainder of this proof.

1. Noticing that $P^{st}(x) | \prod_{i=1,j=1}^{i=n,j=m}(x - \lambda_i - \mu_j)^{\deg(p_i(x))+\deg(q_i(x))}$ as shown in Lemma 2.3 gives $deg^P(s(x)t(x)) \leq deg^P(s(x))deg^P(t(x))$.
2. Observing that $P^{s+t}(x)|P^s(x)P^t(x)$, as shown in Lemma 2.3 gives $deg^P(s(x)+t(x)) \leq deg^P(s(x)) + deg^P(t(x))$. The lower bound comes from taking $s(x) = -t(x)$.
3. In Lemma 2.3 it was shown that $P^{s'}(x) = P^s(x)$ or $xP^{s'}(x) = P^s(x)$. Hence $deg^P(s'(x)) = deg^P(s(x))$ or $deg^P(s(x)) - 1$.
4. In Lemma 2.3 it was shown that $P^{\int_0^x s(y)dy}(x) = P^s(x)$ or $P^{\int_0^x s(y)dy}(x) = xP^s(x)$. Hence $deg^P(\int_0^x s(y)dy) = deg^P(s(x))$ or $deg^P(s(x)) + 1$.
5. If $P^s(x) = \prod_{i=1}^n (x - \lambda_i)^{\deg(p_i(x))}$ then $P^{s(\alpha x)}(x) = \prod_{i=1}^n (x - \alpha\lambda_i)^{\deg(p_i(x))}$, which has the same degree. Hence $deg^P(s(\alpha x)) = deg^P(s(x))$.
6. Part 2 and part 5 of this lemma, in combination shows that $deg^P(\sum_{k=1}^m s(\omega_m^k x)\omega_m^{-qk}) \leq \sum_{k=1}^m deg^P(s(\omega_m^k x)) = m \times deg^P(s(x))$. The lower bound follows by considering the same example as is found in Lemma 2.4 part 6.

■

Chapters 4 and 5 typically work with the recurrences instead of with the poly-exponential function directly. These results are useful as they give bounds for the linear recurrence relations. The bound given by the metric deg^P is obvious, and the metric deg^d gives a bound to the multiplicity of roots in the recurrence polynomial.

Now the relationship between the metrics is examined.

Lemma 2.6 *Let* $s(x) \in \mathcal{P}$. *Then* $1 + deg^d(s(x)) \leq deg^P(s(x))$.

Proof: Write $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$ for the remainder of this proof. By Corollary 1 it follows that:

$$1 + deg^d(s(x)) \quad = \quad \max_{i=1}^{n}(\deg(p_i(x)) + 1) \leq \sum_{i=1}^{n}(\deg(p_i(x)) + 1) = deg^P(s(x)).$$

Which gives the desired result.

■

2.7 Examples.

In this section, three detailed examples are worked out. That of $s(x) = \sum_{i=1}^{n} \alpha_i e^{\lambda_i(x)}$ and $t(x) = e^{\lambda x}p(x)$, and the Chebyshev T polynomials.

Example 9 *Consider $s(x) = \sum_{i=1}^{n} \alpha_i\ e^{\lambda_i(x)}$. Therefore the recurrence polynomial is $P^s(x) = \prod_{i=1}^{n}(x-\lambda_i)$. Denoting $\beta_k = \sum_{J\subseteq\{\lambda_1,\ldots,\lambda_n\},|J|=k} \prod_{\lambda\in J} \lambda$ to be the elementary symmetric polynomials of N variables gives $P^s(x) = \sum_{k=0}^{N} x^{N-k}\beta_k(-1)^k$.*

Writing $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ gives a linear recurrence relation for the b_i namely $b_i = \sum_{k=1}^{N} \beta_k\ b_{i-k}(-1)^k$.

The first N values of the b_i must be determined. Note that $b_i = s^{(i)}(0)$, the i-th derivative of $s(x)$. Also $s(x) = \sum_{i=1}^{n} \alpha_i e^{\lambda_i x}$, so $b_i = \sum_{k=1}^{n} \alpha_k \lambda_k^i$.

Example 10 *Consider $t(x) = e^{\lambda x}p(x)$. So the recurrence polynomial satisfies $P^t(x) = (x-\lambda)^{\deg(p(x))}$. Let $N = \deg(p(x))$ for convenience. Consequently $P^t(x) = \sum_{k=0}^{N} \binom{N}{k} x^{N-k}(-\lambda)^k$. Thus linear recurrence relation is simply: $b_i = -\sum_{k=0}^{N} \binom{N}{k} b_{i-k}(-\lambda)^k$.*

Now determine the first N values of the b_i. Observe that $b_i = t^{(i)}(0)$, the i-th derivative of $t(x)$. Further, observe that $t(x) = e^{\lambda x}p(x)$. So $t^{(0)}(0)$ is simply $p(0)$. Next $t^{(1)}(0) = \lambda p(0) + p'(0)$. Next $t^{(2)}(0) = \lambda^2 p(0) + 2\lambda p'(0) + p''(0)$. In general $t^{(k)}(0) = \sum_{i=0}^{k} \binom{k}{i} \lambda^{k-i} p^{(i)}(0)$. If $p(x) = a_N x^N + ... + a_0$, then this formula for the b_i will simplify to $t^{(k)}(0) = \sum_{i=0}^{k} \binom{k}{i} \lambda^{k-i} i! a_i$.

Thus the linear recurrence relation is $b_i = -\sum_{k=0}^{N} b_{i-k}(-\lambda)^k$ and where for $k < N$, $b_i = \sum_{k=0}^{i} \binom{i}{k} \lambda^{i-k} k! a_k$.

Example 11 *Consider the following example in Maple.*

```
> with(MS):
```

*This example will demonstrate that process of multisectioning can be used where the recurrence has symbolic values rather than simply numeric ones. Consider the "*Chebyshev T polynomials*", as* *polynomials in t, with the recurrence $T_n = 2\,t\,T_{n-1} - T_{n-2}$ with initial polynomials $T_0 = 1$ and $T_1 = t$ [2]. Consider multisectioning this by 5 at 1, to get a recurrence for T_1, T_6, T_{11}, T_{16}, ...*

```
> egf := f(x) = 2*t*f(x-1)-f(x-2),f,x,[f(0)=1, f(1)=t];
```

$$egf := \mathrm{f}(x) = 2\,t\,\mathrm{f}(x-1) - \mathrm{f}(x-2),\ f,\ x,\ [\mathrm{f}(0) = 1,\ \mathrm{f}(1) = t]$$

```
> `egf/ms`(egf,5,1);
```

$$\begin{aligned}&\mathrm{f}(x) = -\mathrm{f}(x-10) + (32\,t^5 - 40\,t^3 + 10\,t)\,\mathrm{f}(x-5),\ f,\ x, [\mathrm{f}(0) = 0,\ \mathrm{f}(1) = t,\ \mathrm{f}(2) = 0,\\ &\quad \mathrm{f}(3) = 0,\ \mathrm{f}(4) = 0,\ \mathrm{f}(5) = 0, \mathrm{f}(6) =\\ &\quad 2\,t\,(2\,t\,(2\,t\,(2\,t\,(2\,t^2 - 1) - t) - 2\,t^2 + 1) - 2\,t\,(2\,t^2 - 1) + t)\\ &\quad -2\,t\,(2\,t\,(2\,t^2 - 1) - t) + 2\,t^2 - 1,\ \mathrm{f}(7) = 0,\ \mathrm{f}(8) = 0,\ \mathrm{f}(9) = 0]\end{aligned}$$

```
> expand([%]);
```

$$\begin{aligned}&[\mathrm{f}(x) = -\mathrm{f}(x-10) + 32\,\mathrm{f}(x-5)\,t^5 - 40\,\mathrm{f}(x-5)\,t^3 + 10\,\mathrm{f}(x-5)\,t,\ f,\ x, [\mathrm{f}(0) = 0,\ \mathrm{f}(1) = t,\\ &\quad \mathrm{f}(2) = 0,\ \mathrm{f}(3) = 0,\ \mathrm{f}(4) = 0,\ \mathrm{f}(5) = 0,\ \mathrm{f}(6) = 32\,t^6 - 48\,t^4 + 18\,t^2 - 1,\ \mathrm{f}(7) = 0,\\ &\quad \mathrm{f}(8) = 0,\ \mathrm{f}(9) = 0]]\end{aligned}$$

This example is interesting in that it shows that the λ_i used in the definition of poly-exponential functions, (Definition 2.1) can be symbolic values in the complex numbers, as opposed to just the numeric values.

2.8 Conclusions.

By combining the results of Theorem 2.1, Lemmas 2.3, 2.5 and Corollary 2 the following results are true.

Theorem 2.3 *Let $s(x) \in \mathcal{P}$.*

1. *Then there exists a lacunary recurrence relation for the mi+q-th coefficient of $s(x)$'s exponential generating function in terms of the $mj+q$-th coefficient $j = i-N, ..., i-1$, where N is bounded above by $deg^P(s(x))$.*

2. *Moreover if the linear recurrence relation associated with $s(x)$ is such that the associated recurrence polynomial is in $R_1[x]$, then the recurrence polynomial of the new lacunary recurrence relation will also be in $R_1[x]$.*

3. *Furthermore if the linear recurrence relation associated with $s(x)$ is of length N, then the new lacunary recurrence relation will be of length less than or equal to mN, where only $\frac{1}{m}$-th of the terms are non-zero.*

The following corollory was know in [16], but its proof was specific to either the Fibonacci or Lucas type I numbers, and was not the consequence of a more general theorem.

Corollary 5 *The $mi + q$ term of the Fibonacci and Lucas type I numbers can be computed in terms of $mj+q$ term for $j = i-2, i-1$ via a lacunary recurrence relation. Moreover the lacunary recurrence relation will be over $\mathbb{Z}$. Lastly, the lacunary recurrence relation will be of length $2m$ with 2 non-zero terms.*

Chapter 3

Rational poly-exponential functions.

3.1 Rational poly-exponential function.

Some techniques for poly-exponential functions where developed in Chapter 2. This chapter expands the scope of the study to a more general setting; that of ratios of poly-exponential functions. To that end, define:

Definition 3.1 (Rational poly-exponential function.) *Let $s(x)$, $t(x) \in \mathcal{P}$ and $t(x) \neq 0$. Then*

$$\frac{s(x)}{t(x)},$$

*is a "*rational poly-exponential function*". Denote the set of all such functions by $\mathcal{R}$.*

This definition was suggested by my supervisor, Jon Borwein, as a generalization of the Bernoulli numbers. All of the methods Lehmer, or Glaisher [19, 14] to multisectioning the Bernoulli numbers relied only upon the fact that these numbers had "nice" linear recurrence relation to describe the exponential generating function of the numerator and denominator. Definition 3.1 maintains this property, but expands the scope of the results to a much large class of functions. To the best of my knowledge, the results in this chapter are new, in the sense that they have not been done in this degree of generality before.

Section 3.2 shows how to calculate the coefficients of the exponential generating function of functions in $\mathcal{R}$ by use of recursion formulae. Section 3.3 will demonstrate the effects of multisectioning on functions in $\mathcal{R}$. The structure of $\mathcal{R}$ is studied in Section 3.4, examining different rings, subrings, fields and subfields of $\mathcal{R}$, along with some closure properties. In Section 3.5 the examination of subfields of $\mathcal{R}$ is continued, by exploring how these subfields relate to each other. Some metrics of complexity for functions in $\mathcal{R}$ are investigated in Section 3.6. Section 3.7 contains three worked out examples. The last section, Section 3.8 summarizes the main points of this chapter into a final theorem.

3.2 Recursion formula for functions in $\mathcal{R}$.

The study of rational poly-exponential functions begins by looking at an example of how to calculate the coefficients of the exponential generating function of $\frac{x}{e^x-1}$. These are the "*Bernoulli numbers*" (in even suffix notation) [2].

Example 12 *Define*

$$\sum_{k=0}^{\infty} c_k \frac{x^k}{k!} = \frac{x}{e^x-1} = \frac{\sum_{i=0}^{\infty} b_i \frac{x^i}{i!}}{\sum_{j=0}^{\infty} d_j \frac{x^j}{j!}}.$$

Then the c_k are the Bernoulli numbers. A simple calculation shows that $b_i = 1$ if $i = 1$ and 0 otherwise. Further $d_j = 0$ if $j = 0$ and 1 otherwise.

Now:

$$\begin{aligned}
\sum_{k=0}^{\infty} c_k \frac{x^k}{k!} &= \frac{\sum_{i=0}^{\infty} b_i \frac{x^i}{i!}}{\sum_{j=0}^{\infty} d_j \frac{x^j}{j!}} \\
\sum_{j=0}^{\infty} d_j \frac{x^j}{j!} \sum_{k=0}^{\infty} c_k \frac{x^k}{k!} &= \sum_{i=0}^{\infty} b_i \frac{x^i}{i!} \\
\sum_{k=0}^{\infty} \sum_{j=0}^{k} d_j \frac{x^j}{j!} c_{k-j} \frac{x^{k-j}}{(k-j)!} &= \sum_{i=0}^{\infty} b_i \frac{x^i}{i!} \\
\sum_{k=0}^{\infty} \sum_{j=0}^{k} \binom{k}{j} d_j c_{k-j} \frac{x^k}{k!} &= \sum_{i=0}^{\infty} b_i \frac{x^i}{i!} \\
\sum_{j=0}^{k} \binom{k}{j} d_j c_{k-j} &= b_k.
\end{aligned}$$

From this a recursion formula for the Bernoulli numbers is derived that, for $k > 2$ gives:

$$\sum_{j=1}^{k} \binom{k}{j} c_{k-j} = 0$$

$$c_{k-1} = \frac{-1}{k}\sum_{j=0}^{k-2}\binom{n}{j}c_j.$$

This is the standard recursion formula used for the Bernoulli numbers, as would be found in [2, 10, 16].

Note 3.1 *It is important to note that the term "linear recurrence relation" is different than that of "recursion formula". A recursion formula is a formula where the n-th term depends on the previous $n-1$ terms, where as a linear recurrence relation only requires the previous N terms of which a linear combination is used to determine the n-th term. Examples 12 gives a recursion formula for the Bernoulli numbers.*

It is not always possible to write $f(x) \in \mathcal{R}$ as $\sum_{i=0}^{\infty} c_i \frac{x^i}{i!}$. In particular if $f(x)$ has a pole at 0, this will not be possible (i.e. $\frac{1}{x}$). The restriction to $f(x) \in \mathcal{R}$ which do not have poles at 0, is closed under addition, differentiation, multiplication, $f(x) \to f(\alpha x)$ and multisectioning, by simply looking at the Taylor series under these operations. Denote this set as $\hat{\mathcal{R}}$ to get this definition:

Definition 3.2 ($\hat{\mathcal{R}}$.) *Define*

$$\hat{\mathcal{R}} = \{f(x) : \lim_{x\to 0}\frac{1}{f(x)} \neq 0, f(x) \in \mathcal{R}\}.$$

3.3 Multisectioning.

This section explores the effects of multisectioning on rational poly-exponential functions. The main result of this section allows for the improvement in the efficiency of calculating the coefficients of exponential generating functions for functions in $\mathcal{R}$.

Lemma 3.1 *If $h(x) \in \mathcal{R}$ then $h_m^q(x)$ can be written as $\frac{s_m^q(x)}{t_m^0(x)}$ where $s(x)$, $t(x) \in \mathcal{P}$.*

Proof: Write $h(x) = \frac{s_h(x)}{t_h(x)}$. Thus:

$$\begin{aligned} h_m^q(x) &= \frac{1}{m}\sum_{i=0}^{m-1}\frac{\omega_m^{-iq}s_h(x\omega_m^i)}{t_h(x\omega_m^i)} = \frac{1}{m}\sum_{i=1}^{m-1}\frac{\omega_m^{-iq}s_h(x\omega_m^i)\prod_{j=1}^{m-1}t_h(x\omega_m^{j+i})}{\prod_{j=0}^{m-1}t_h(x\omega_m^j)} \\ &= \frac{\frac{1}{m}\sum_{i=1}^{m-1}\omega_m^{-iq}s_h(x\omega_m^i)\prod_{j=1}^{m-1}t_h(x\omega_m^{j+i})}{\prod_{j=0}^{m-1}t_h(x\omega_m^j)} = \frac{(s_h(x)\prod_{j=1}^{m-1}t_h(x\omega_m^j))_m^q}{(\prod_{j=0}^{m-1}t_h(x\omega_m^j))_m^0}. \end{aligned}$$

Picking $s(x) = s_h(x)\prod_{i=1}^{m-1}t_h(x\omega_m^i)$ and $t(x) = \prod_{i=0}^{m-1}t_h(x\omega_m^i)$ gives the desired result. It is also worthwhile to note that $t_m^0(x) = t(x)$.

■

Theorem 3.1 *Given a function $f(x) \in \hat{\mathcal{R}}$, $m, q \in \mathbb{Z}$, $0 \leq q < m$, a recursion formula can be found for the $mi+q$-th coefficient of the exponential generating function of $f(x)$ that depends only on the $mj+q$-th coefficient, for $j < i$, and two lacunary recurrence relations.*

Later, in Section 3.8, by combining this theorem, Theorem 3.1, with some later results, Lemmas 3.3, 3.4 and 3.6, an even tighter result will be given, the lengths of these lacunary recurrence relations, will be determined, and the ring that their coefficients will lie will be known.

Proof: Let

$$f(x) = \sum_{i=0}^{\infty} c_i \frac{x^i}{i!} = \frac{s_f(x)}{t_f(x)} = \frac{\sum_{i=0}^{\infty} b_i \frac{x^i}{i!}}{\sum_{j=0}^{\infty} d_j \frac{x^j}{j!}},$$

where $s(x)$, $t(x) \in \mathcal{P}$. Lemma 3.1 gives

$$f_m^q(x) = \sum_{i=0}^{\infty} c_{mi+q} \frac{x^{mi+q}}{(mi+q)!} = \frac{s_m^q(x)}{t_m^0(x)} = \frac{\sum_{i=0}^{\infty} \bar{b}_{mi+q} \frac{x^{mi+q}}{(mi+q)!}}{\sum_{j=0}^{\infty} \bar{d}_{mj} \frac{x^{mj}}{(mj)!}},$$

where s_m^q, $t_m^0 \in \mathcal{P}$, and the $\bar{b}_i$ and the $\bar{d}_j$ satisfy lacunary recurrence relations.

A simple calculation shows that

$$\begin{aligned}
\sum_{i=0}^{\infty} c_{mi+q} \frac{x^{mi+q}}{(mi+q)!} &= \frac{\sum_{i=0}^{\infty} \bar{b}_{mi+q} \frac{x^{mi+q}}{(mi+q)!}}{\sum_{j=0}^{\infty} \bar{d}_{mj} \frac{x^{mj}}{(mj)!}} \\
\sum_{j=0}^{\infty} \bar{d}_{mj} \frac{x^{mj}}{(mj)!} \sum_{i=0}^{\infty} c_{mi+q} \frac{x^{mi+q}}{(mi+q)!} &= \sum_{i=0}^{\infty} \bar{b}_{mi+q} \frac{x^{mi+q}}{(mi+q)!} \\
\sum_{i=0}^{\infty} \sum_{j=0}^{i} \binom{mi+q}{mj} \bar{d}_{mj} c_{m(i-j)+q} \frac{x^{mi+q}}{(mi+q)!} &= \sum_{i=0}^{\infty} \bar{b}_{mi+q} \frac{x^{mi+q}}{(mi+q)!} \\
\sum_{j=0}^{i} \binom{mi+q}{mj} \bar{d}_{mj} c_{m(i-j)+q} &= \bar{b}_{mi+q}.
\end{aligned}$$

Picking $s = \min\{j : d_{mj} \neq 0\}$ gives:

$$\begin{aligned}
\sum_{j=s}^{i} \binom{mi+q}{mj} \bar{d}_{mj} c_{m(i-j)+q} &= \bar{b}_{mi+q} \\
\binom{mi+q}{ms} \bar{d}_{ms} c_{m(i-s)+q} &= \bar{b}_{mi+q} - \sum_{j=s+1}^{i} \binom{mi+q}{mj} \bar{d}_{mj} c_{m(i-j)+q} \\
c_{m(i-s)+q} &= \frac{1}{\binom{mi+q}{ms} \bar{d}_{ms}} (\bar{b}_{mi+q} - \sum_{j=s+1}^{i} \binom{mi+q}{mj} \bar{d}_{mj} c_{m(i-j)+q}).
\end{aligned}$$

Let $k = i - s$, to get

$$\begin{aligned} c_{mk+q} &= \frac{1}{\binom{m(s+k)+q}{ms}\bar{d}_{ms}}(\bar{b}_{m(k+s)+q} - \sum_{j=s+1}^{k+s}\binom{m(k+s)+q}{mj}\bar{d}_{mj}c_{m((k+s)-j)+q}) \\ &= \frac{1}{\binom{m(s+k)+q}{ms}\bar{d}_{ms}}(\bar{b}_{m(k+s)+q} - \sum_{j=1}^{k}\binom{m(k+s)+q}{m(j+s)}\bar{d}_{m(j+s)}c_{m(k-j)+q}). \end{aligned}$$

This is a recursion formula for the c_{mk+q} based on the previous c_{mj+q} with $j < k$ and two lacunary recurrence relations for the $\bar{b}_{mi+q}$ and $\bar{d}_{mi}$.

■

The recursion formula associated with $f_m^q(x)$ is called the "*lacunary recursion formula*" [8, 14].

Example 13 *Consider the following example in Maple. For more information about the Maple code, see Appendix A. For the Maple code see Appendix E. The Maple code and help files (including information about syntax) are available on the web at [1].*

```
> with(MS):
```

Consider again the Bernoulli numbers $\frac{x}{e^x-1} = \frac{\sum_{i=0}^{\infty}\frac{b_i x^i}{i!}}{\sum_{j=0}^{\infty}\frac{d_j x^j}{j!}}$. *Multisection this by 3 at 1, using the formula, as given in Lemma 3.1. After this, this example will calculate the 1-st, 4-th 7-th and 10-th Bernoulli number, using the formula given in Theorem 3.1.*

Let $s_h(x) = x$ *and* $t_h(x) = e^x - 1$, *and solve for* $\mathrm{s}(x)$ *and* $\mathrm{t}(x)$ *in the theorem.*

```
> s[h] := x -> x;
```

$$s_h := x \to x$$

```
> t[h] := (x) -> exp(x)-1;
```

$$t_h := x \to e^x - 1$$

```
> omega[3] := exp(2*Pi*I/3);
```

$$\omega_3 := -\frac{1}{2} + \frac{1}{2} I\sqrt{3}$$

From Lemma 3.1 $\mathrm{s}(x) = s_h(x)\,(\prod_{i=1}^{m-1} t_h\,\omega_m{}^i)$, *and* $\mathrm{t}(x) = \prod_{i=0}^{m-1} t_h(x\,\omega_m{}^i)$, *which, for this particular case is:*

```
> S := s[h](x) * t[h](x*omega[3]) * t[h](x*omega[3]^2);
```

$$S := x\,(e^{(x\,(-1/2+1/2\,I\,\sqrt{3}))} - 1)\,(e^{(x\,(-1/2+1/2\,I\,\sqrt{3})^2)} - 1)$$

```
> T := t[h](x)*t[h](x*omega[3])*t[h](x*omega[3]^2);
```

$$T := (e^x - 1)\,(e^{(x\,(-1/2+1/2\,I\,\sqrt{3}))} - 1)\,(e^{(x\,(-1/2+1/2\,I\,\sqrt{3})^2)} - 1)$$

Now, determine what the linear recurrence relation for this would be.

```
> `pe/ms`(S,b,x,3,1);
```

$$\mathrm{b}(x) = -\mathrm{b}(x-12) + 2\,\mathrm{b}(x-6),\ b,\ x, [\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 0,\ \mathrm{b}(2) = 0,\ \mathrm{b}(3) = 0,\ \mathrm{b}(4) = -12,\ \mathrm{b}(5) = 0,\ \mathrm{b}(6) = 0,\ \mathrm{b}(7) = -7,\ \mathrm{b}(8) = 0,\ \mathrm{b}(9) = 0,\ \mathrm{b}(10) = -30,\ \mathrm{b}(11) = 0,\ \mathrm{b}(12) = 0,\ \mathrm{b}(13) = -13]$$

So $s_3^1(x) = \sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}$, *where* $b_i = b_{i-12} + 2\, b_{i-6}$, *with initial values of* $b_4 = -12$, $b_7 = -7$, $b_{10} = -30$ *and* $b_{13} = -13$.

```
> convert_egf(T,d,x);
```

$$\mathrm{d}(x) = \mathrm{d}(x-6),\ d,\ x,\ [\mathrm{d}(0) = 0,\ \mathrm{d}(1) = 0,\ \mathrm{d}(2) = 0,\ \mathrm{d}(3) = 6,\ \mathrm{d}(4) = 0,\ \mathrm{d}(5) = 0]$$

So the bottom linear recurrence relation $t_3^0(x) = \sum_{j=0}^{\infty} \frac{d_j\, x^i}{j!}$, *where* $d_j = d_{j-6}$, *and the initial values are* $d_3 = 6$.

Equally easy the two built-in commands could have been used to do this in the naive fashion.

```
> top := `top/ms/naive`(x,exp(x)-1,b,x,3,1);
```

$$top := \mathrm{b}(x) = -\mathrm{b}(x-12) + 2\,\mathrm{b}(x-6),\ b,\ x, [\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 0,\ \mathrm{b}(2) = 0,\ \mathrm{b}(3) = 0,\ \mathrm{b}(4) = -12,\ \mathrm{b}(5) = 0,\ \mathrm{b}(6) = 0,\ \mathrm{b}(7) = -7,\ \mathrm{b}(8) = 0,\ \mathrm{b}(9) = 0,\ \mathrm{b}(10) = -30,\ \mathrm{b}(11) = 0,\ \mathrm{b}(12) = 0,\ \mathrm{b}(13) = -13]$$

```
> bot := `bottom/ms/naive`(exp(x)-1,d,x,3);
```

$$bot := \mathrm{d}(x) = \mathrm{d}(x-6),\ d,\ x,\ [\mathrm{d}(0) = 0,\ \mathrm{d}(1) = 0,\ \mathrm{d}(2) = 0,\ \mathrm{d}(3) = 6,\ \mathrm{d}(4) = 0,\ \mathrm{d}(5) = 0]$$

Now, to calculate the first few Bernoulli numbers, use the formula as given in Theorem 3.1, first noting that s is equal to 1.

```
> Top := `egf/makeproc`(top):

> Bot := `egf/makeproc`(bot):

> s := 1:

> m := 3:

> k := 0:

> q := 1:

> Bernoulli[m * k + q] := 1/binomial(m*(s + k) + q, m * s) / Bot(m * s) *
>  (Top(m *(k + s) + q) - add(binomial (m *(k + s) + q,
>  m *(j + s)) * Bot(m * j) * Bernoulli[m * (k+s-j) + q],
>  j = 1+s ..  k+s));
```

$$Bernoulli_1 := \frac{-1}{2}$$

```
>  k := 1:
>  Bernoulli[m * k + q] := 1/binomial(m*(s + k) + q, m * s) / Bot(m * s) *
>  (Top(m *(k + s) + q) - add(binomial (m *(k + s) + q,
>  m *(j + s)) * Bot(m * (j + s)) * Bernoulli[m * (k-j) + q],
>  j = 1 ..  k));
```

$$Bernoulli_4 := \frac{-1}{30}$$

```
>  k := 2:
>  Bernoulli[m * k + q] := 1/binomial(m*(s + k) + q, m * s) / Bot(m * s) *
>  (Top(m *(k + s) + q) - add(binomial (m *(k + s) + q,
>  m *(j + s)) * Bot(m * (j + s)) * Bernoulli[m * (k-j) + q],
>  j = 1 ..  k));
```

$$Bernoulli_7 := 0$$

```
>  k := 3:
>  Bernoulli[m * k + q] := 1/binomial(m*(s + k) + q, m * s) / Bot(m * s) *
>  (Top(m *(k + s) + q) - add(binomial (m* (k + s) + q,
>  m *(j + s)) * Bot(m * (j + s)) * Bernoulli[m * (k-j) + q],
>  j = 1 ..  k));
```

$$Bernoulli_{10} := \frac{5}{66}$$

There is automated code to get the same result.

```
>  A := `calcul/normal`(10, Top, Bot, 3, 1):
>  seq(A[3 * i + 1], i = 0 ..3);
```

$$\frac{-1}{2}, \frac{-1}{30}, 0, \frac{5}{66}$$

3.4 The structure of $\mathcal{R}$.

Like $\mathcal{P}$, this section will show that $\mathcal{R}$ has a rich structure. To explore this structure, this section first makes some definitions for subsets of $\mathcal{R}$ analogous to the Definitions 2.3 and 2.4 for $\mathcal{P}$.

Definition 3.3 **($\mathcal{R}^{R_1,R_2}$, $\mathcal{R}_{R_1,R_2}$.)** *Let R_1 and R_2 be subrings of $\mathbb{C}$. Denote $\mathcal{R}^{R_1,R_2}$ ($\mathcal{R}_{R_1,R_2}$) to be the subset of $\mathcal{R}$, such that all elements can be written in for the form $\frac{s(x)}{t(x)}$ with $s(x)$, $t(x) \in \mathcal{P}^{R_1,R_2}$ ($s(x)$, $t(x) \in \mathcal{P}_{R_1,R_2}$).*

Definition 3.4 **($\hat{\mathcal{R}}^{R_1,R_2}$, $\hat{\mathcal{R}}_{R_1,R_2}$.)** *Let R_1 and R_2 be subrings of $\mathbb{C}$. Define $\hat{\mathcal{R}}^{R_1,R_2} = \mathcal{R}^{R_1,R_2} \cap \hat{\mathcal{R}}$ and $\hat{\mathcal{R}}_{R_1,R_2} = \mathcal{R}_{R_1,R_2} \cap \hat{\mathcal{R}}$.*

First collect some closure properties for $\mathcal{R}$.

Lemma 3.2 *Let R_1, R_2, R_3, and R_4 be subrings of $\mathbb{C}$ and let $h(x) \in \mathcal{R}_{R_1,R_2}$ and $g(x) \in \mathcal{R}_{R_3,R_4}$ then:*

1. *$g(x)h(x) \in \mathcal{R}_{\langle R_1,R_3\rangle,R_2R_4}$,*
2. *$g(x) + h(x) \in \mathcal{R}_{\langle R_1,R_3\rangle,R_2R_4}$,*
3. *$h'(x) \in \mathcal{R}_{R_1,\langle R_1,R_2\rangle}$,*
4. *$h^q_m(x) \in \mathcal{R}_{R_1\langle\omega_m\rangle,R_2\langle\omega_m\rangle}$.*

Proof: For convenience, write $h(x) = \frac{s_h(x)}{t_h(x)}$, with $s_h(x)$, $t_h(x) \in \mathcal{P}_{R_1,R_2}$, and $g(x) = \frac{s_g(x)}{t_g(x)}$, with $s_g(x)$, $t_g(x) \in \mathcal{P}_{R_3,R_4}$.

1. Now $g(x)h(x) = \frac{s_g(x)s_h(x)}{t_g(x)t_h(x)}$, so by Lemma 2.2 it follows that $s_g(x)s_h(x) \in \mathcal{P}_{\langle R_1,R_3\rangle,R_2R_4}$, and $t_g(x)t_h(x) \in \mathcal{P}_{\langle R_1,R_3\rangle,R_2R_4}$. Consequently $g(x)h(x) \in \mathcal{R}_{\langle R_1,R_3\rangle,R_2R_4}$.
2. Observe that $g(x) + h(x) = \frac{s_h(x)t_g(x)+s_g(x)t_h(x)}{t_g(x)t_h(x)}$. From Lemma 2.2 $s_g(x)t_h(x) + t_g(x)s_h(x) \in \mathcal{P}_{\langle R_1,R_3\rangle,R_2R_4}$, and $t_g(x)t_h(x) \in \mathcal{P}_{\langle R_1,R_3\rangle,R_2R_4}$. Hence $g(x) + h(x) \in \mathcal{R}_{\langle R_1,R_3\rangle,R_2R_4}$.
3. By considering $h'(x) = \frac{s'_h(x)t_h(x)-s_h(x)t'_h(x)}{t^2_h(x)}$, and Lemma 2.2 it is seen that $s'_h(x)t_h(x) - t'_h(x)s_h(x) \in \mathcal{P}_{R_1,\langle R_1,R_2\rangle}$ and $t^2_h(x) \in \mathcal{P}_{R_1,R_2}$. Thus $h'(x) \in \mathcal{R}_{R_1,\langle R_1,R_2\rangle}$.
4. Now $h^q_m(x) = \frac{(s_h(x)\prod_{i=1}^{m-1} t_h(x\omega^i_m))^q_m}{(\prod_{i=0}^{m-1} t_h(x\omega^i_m))^0_m}$ (Lemma 3.1). From Lemma 2.2 the numerator and the denominator are both in $\mathcal{P}_{R_1\langle\omega_m\rangle,R_2\langle\omega_m\rangle}$. This gives $h^q_m(x) \in \mathcal{R}_{R_1\langle\omega_m\rangle,R_2\langle\omega_m\rangle}$.

■

Lemma 3.3 *Let R_1, R_2, R_3, and R_4 be subrings of $\mathbb{C}$ and let $h(x) \in \mathcal{R}^{R_1,R_2}$ and $g(x) \in \mathcal{R}^{R_3,R_4}$ then:*

1. $g(x)h(x) \in \mathcal{R}^{\langle R_1,R_3\rangle, R_2R_4}$,
2. $g(x) + h(x) \in \mathcal{R}^{\langle R_1,R_3\rangle, R_2R_4}$,
3. $h'(x) \in \mathcal{R}^{R_1,R_2}$,
4. $h^q_m(x) \in \mathcal{R}^{R_1\langle\omega_m\rangle,R_2}$.

Proof: For convenience, write $h(x) = \frac{s_h(x)}{t_h(x)}$, with $s_h(x)$, $t_h(x) \in \mathcal{P}^{R_1,R_2}$, and $g(x) = \frac{s_g(x)}{t_g(x)}$, with $s_g(x)$, $t_g(x) \in \mathcal{P}^{R_3,R_4}$.

1. As $g(x)h(x) = \frac{s_g(x)s_h(x)}{t_g(x)t_h(x)}$ and Lemma 2.3 it follows that $s_g(x)s_h(x) \in \mathcal{P}^{\langle R_1,R_3\rangle,R_2R_4}$, and $t_g(x)t_h(x) \in \mathcal{P}^{\langle R_1,R_3\rangle,R_2R_4}$. Consequently $g(x)h(x) \in \mathcal{R}^{\langle R_1,R_3\rangle,R_2R_4}$.
2. Observing that $g(x)+h(x) = \frac{s_h(x)t_g(x)+s_g(x)t_h(x)}{t_g(x)t_h(x)}$, and appealing to Lemma 2.3 gives $s_g(x)t_h(x) + t_g(x)s_h(x) \in \mathcal{P}^{\langle R_1,R_3\rangle,R_2R_4}$ and $t_g(x)t_h(x) \in \mathcal{P}^{\langle R_1,R_3\rangle,R_2R_4}$. Hence $h(x)+g(x) \in \mathcal{R}^{\langle R_1,R_3\rangle,R_2R_4}$.
3. Now $h'(x) = \frac{s'_h(x)t_h(x)-s_h(x)t'_h(x)}{t^2_h(x)}$. So from Lemma 2.3 it follows that $s'_h(x)t_h(x) - t'_h(x)s_h(x) \in \mathcal{P}^{R_1,R_2}$ and $t^2_h(x) \in \mathcal{P}^{R_1,R_2}$. Thus $h'(x) \in \mathcal{R}^{R_1,R_2}$.
4. As a result of $h^q_m(x) = \frac{(s_h(x)\prod_{i=1}^{m-1} t_h(x\omega_m^i))^q_m}{(\prod_{i=0}^{m-1} t_h(x\omega_m^i))^0_m}$ (Lemma 3.1), and Lemma 2.3 it follows that both the numerator and the denominator are in $\mathcal{P}^{\langle\omega_m\rangle R_1,\langle\omega_m\rangle R_2}$. A tighter bound on the denominator $\prod_{i=0}^{m-1} t_h(x\omega_m^i)$ and numerator $(s_h(x)\prod_{i=1}^{m-1} t_h(x\omega_m^i))^q_m$, by noticing that they are fixed by automorphism of the number field $\langle\omega\rangle$ and hence are in $\mathcal{P}^{\langle\omega_m\rangle R_1,R_2}$.

■

Corollary 6 *Let R_1 and R_2 be subrings of $\mathbb{C}$. Then $\mathcal{R}^{R_1,R_2}$ and $\mathcal{R}_{R_1,R_2}$ are both fields, more over $\mathcal{R}^{R_1,R_2}$ is closed under differentiation.*

Corollary 7 *Let R_1 and R_2 be subrings of $\mathbb{C}$. Then $\hat{\mathcal{R}}^{R_1,R_2}$ and $\hat{\mathcal{R}}_{R_1,R_2}$ are both rings, more over $\hat{\mathcal{R}}^{R_1,R_2}$ is closed under differentiation.*

Now examine some closure properties of the recurrence polynomial.

Lemma 3.4 *Assume that $h(x)$, $g(x) \in \mathcal{R}$, where $h(x) = \frac{s_h(x)}{t_h(x)}$ and $g(x) = \frac{s_g(x)}{t_g(x)}$ with $s_h(x)$, $t_h(x)$, $s_g(x)$, $t_g(x) \in \mathcal{P}$. Let R_1, R_2, R_3 and R_4 be subrings of $\mathbb{C}$ and assume that $P^{s_h}(x) \in R_1[x]$, $P^{t_h}(x) \in R_2[x]$, $P^{s_g}(x) \in R_3[x]$ and $P^{t_g}(x) \in R_4[x]$.*

1. *Then* $g(x)h(x) = \frac{s_{gh}(x)}{t_{gh}(x)}$, *where* $P^{s_{gh}}(x) \in \langle R_1, R_3\rangle[x]$ *and* $P^{t_{gh}}(x) \in \langle R_2, R_4\rangle[x]$.
2. *Then* $g(x) + h(x) = \frac{s_{g+h}(x)}{t_{g+h}(x)}$, *where* $P^{s_{g+h}}(x) \in \langle R_1, R_2, R_3, R_4\rangle[x]$ *and* $P^{t_{g+h}}(x) \in \langle R_2, R_4\rangle[x]$.
3. *Then* $h'(x) = \frac{s_{h'}(x)}{t_{h'}(x)}$, *where* $P^{s_{h'}}(x) \in \langle R_1, R_2\rangle[x]$ *and* $P^{t_{h'}}(x) \in R_2[x]$.
4. *Then* $h^q_m(x) = \frac{s_{h^q_m}(x)}{t_{h^q_m}(x)}$, *where* $P^{s_{h^q_m}}(x) \in \langle R_1, R_2\rangle[x]$ *and* $P^{t_{h^q_m}}(x) \in R_2[x]$.

Proof:

1. By letting $s_{gh}(x) = s_g(x)s_h(x)$ and $t_{gh}(x) = t_g(x)t_h(x)$ the result follows from Corollary 2.
2. By letting $s_{g+h}(x) = s_g(x)t_h(x) + s_h(x)t_g(x)$ and $t_{g+h}(x) = t_g(x)t_h(x)$ the result follows from Corollary 2.
3. By letting $s_{h'}(x) = s'_h(x)t_h(x) - s_h(x)t'_h(x)$ and $t_{h'}(x) = t^2_h(x)$ the result follows from Corollary 2.
4. By letting $s_{h^q_m}(x) = (s_h(x)\prod_{i=1}^{m-1} t_h(x\omega^i_m))^q_m$ and $t_{h^q_m}(x) = (\prod_{i=0}^{m-1} t_h(x\omega^i_m))^0_m$ the result follows from Corollary 2.

$\blacksquare$

These results are useful, as they allow the assumption to be made that certain calculations will always be over nice rings, (for example, the lacunary recurrence relation for the Euler numbers will be over the integers).

3.5 Hierarchy of $\mathcal{R}$.

As with $\mathcal{P}$, there is an interrelationship between the different subfields and subrings of $\mathcal{R}$, and a hierarchy of the different subfields.

Theorem 3.2 (Hierarchy.) *If* R_1 *and* R_2 *are subrings of* $\mathbb{C}$ *then the following subset relationships hold:*

1. $\hat{\mathcal{R}}_{R_1,R_2} \subsetneq \mathcal{R}_{R_1,R_2} \subseteq \mathcal{R}^{R_1,R_1R_2}$,
2. $\hat{\mathcal{R}}^{R_1,R_2} \subsetneq \mathcal{R}^{R_1,R_2} \subseteq \mathcal{R}_{R_1,R_1R_2}$.

Proof:

1. If $f(x) \in \mathcal{R}_{R_1,R_2}$, then $f(x) = \frac{s_f(x)}{t_f(x)}$, where $s_f(x),\ t_f(x) \in \mathcal{P}_{R_1,R_2}$, then $s_f(x),\ t_f(x) \in \mathcal{P}^{R_1,\langle R_1,R_1R_2\rangle}$. Take any non-zero element of R_2, say β, and notice that $\beta s_f(x),\ \beta t_f(x) \in \mathcal{P}^{R_1,R_1R_2}$, thus $f(x) = \frac{\beta s_f(x)}{\beta t_f(x)} \in \mathcal{R}^{R_1,R_1R_2}$ as required.

   Noticing that $\hat{\mathcal{R}}_{R_1,R_2} \subsetneq \mathcal{R}_{R_1,R_2}$ follows from noticing that $\hat{\mathcal{R}}_{R_1,R_2}$ is not closed under division.

2. If $f(x) \in \mathcal{R}^{R_1,R_2}$, where $f(x) = \frac{s_f(x)}{t_f(x)}$, with $s_f(x),\ t_s(x) \in \mathcal{P}^{R_1,R_2}$, then $s_f(x),\ t_f(x) \in \mathcal{P}_{R_1,R_2\langle R_1,R_1^{-1}\rangle}$ by Theorem 2.2. Say $s_f(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$, and $t_f(x) = \sum_{j=1}^{m} q_j(x)e^{\mu_j x}$, with $p_i(x), q_i(x) \in R_2\langle R_1, R_1^{-1}\rangle$. For each coefficient of $p_i(x)$ and $q_i(x)$, multiply the coefficient by some $\alpha_i \in R_1$ (dependent on $p_i(x)$) so that the resulting coefficients are in R_1R_2. Now taking the least common multiple of all these α_i, gives some $\beta \in R_1$ such that $\beta p_i(x), \beta q_i(x) \in R_1R_2[x]$ for all i. Then write this as $f(x) = \frac{s_f(x)}{t_f(x)} = \frac{\beta s_f(x)}{\beta t_f(x)}$, where $\beta s_f(x), \beta t_f(x) \in \mathcal{P}_{R_1,R_1R_2}$. Hence $f(x) \in \mathcal{R}_{R_1,R_1R_2}$.

   Noticing that $\hat{\mathcal{R}}^{R_1,R_2} \subsetneq \mathcal{R}^{R_1,R_2}$ follows from noticing that $\hat{\mathcal{R}}^{R_1,R_2}$ is not closed under inversion.

■

Corollary 8 *Let R_1 and R_2 be subrings of $\mathbb{C}$. If $1 \in R_1 \subseteq R_2$ then $\mathcal{R}^{R_1,R_2} = \mathcal{R}_{R_1,R_2}$.*

The next two examples show that the set of rings $\mathcal{R}^{R_1,R_2}$ and that of $\mathcal{R}_{R_1,R_2}$ share neither a superset nor a subset relationship with each other. These examples are such that $\mathcal{R}^{R_1,R_2}$ for particular R_1 and R_2 that cannot be written as $\mathcal{R}_{R_3,R_4}$ for any R_3 and R_4 and vice-versa.

Example 14 *Let $f(x) = e^{\sqrt{2}x} \in \mathcal{R}_{\mathbb{Q}(\sqrt{2}),\mathbb{Q}}$. Notice that $f'(x) = \sqrt{2}e^{\sqrt{2}x} \notin \mathcal{R}_{\mathbb{Q}(\sqrt{2}),\mathbb{Q}}$. But $\mathcal{R}^{R_1,R_2}$ is closed under differentiation. Consequently there do not exist rings R_1, R_2 such that $\mathcal{R}_{\mathbb{Q}(\sqrt{2}),\mathbb{Q}} = \mathcal{R}^{R_1,R_2}$.*

Example 15 *The goal here is to show that there do not exist subrings R_1 and R_2 of $\mathbb{C}$ such that $\mathcal{R}^{\mathbb{Z}[\sqrt{2}],\mathbb{Q}} = \mathcal{R}_{R_1,R_2}$. Consider $s_c(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ where b_i satisfies $b_i = 2c^2 b_{i-2}$ for $c \in \mathbb{Z}$, with $b_0, b_1 \in \mathbb{Z}$. Then $s_c(x) \in \mathcal{R}^{\mathbb{Z}[\sqrt{2}],\mathbb{Q}}$. Further this is equivalent to*

$$s_c(x) = \alpha_1 e^{c\sqrt{2}x} + \alpha 2 e^{-c\sqrt{2}x},$$

where $\alpha_1 = \frac{b_0}{2} + \frac{b_1}{2\sqrt{2}c}$ and $\alpha_2 = \frac{b_0}{2} - \frac{b_1}{2\sqrt{2}c}$. From this conclude that if $\mathcal{R}^{\mathbb{Z}[\sqrt{2}],\mathbb{Q}} = \mathcal{R}_{R_1,R_2}$, then $\mathcal{R}_{\mathbb{Z}[\sqrt{2}],\mathbb{Q}[\sqrt{2}]} \subseteq \mathcal{R}_{R_1,R_2}$.

Observing that $\mathcal{R}^{\mathbb{Z}[\sqrt{2}],\mathbb{Q}[\sqrt{2}]} = \mathcal{R}^{\mathbb{Z}[\sqrt{2}],\mathbb{Q}[\sqrt{2}]} \neq \mathcal{R}^{\mathbb{Z}[\sqrt{2}],\mathbb{Q}}$, as $\sqrt{2} \in \mathcal{R}^{\mathbb{Z}[\sqrt{2}],\mathbb{Q}[\sqrt{2}]}$ and $\sqrt{2} \notin \mathcal{R}^{\mathbb{Z}[\sqrt{2}],\mathbb{Q}}$, gives that $\mathcal{R}^{\mathbb{Z}\langle\sqrt{2}\rangle,\mathbb{Q}} \neq \mathcal{R}_{\mathbb{Z}[\sqrt{2}],\mathbb{Q}[\sqrt{2}]}$ from which the desired result follows.

3.6 Some complexity bounds.

This section determines some metrics of complexity of functions in $\mathcal{R}$, as was done earlier for functions in $\mathcal{P}$ (Section 2.6). This section uses the metrics from Definition 2.7 on the numerator and denominator of functions in $\mathcal{R}$ to get the following lemmas:

Lemma 3.5 (*deg^d*.) *Let $h(x) = \frac{s_h(x)}{t_h(x)}$, $g(x) = \frac{s_g(x)}{t_g(x)} \in \mathcal{R}$, such that $s_h(x)$, $t_h(x)$, $s_g(x)$, $t_g(x) \in \mathcal{P}$. Then:*

1. *Then $f(x) = \frac{s_f(x)}{t_f(x)} = g(x)h(x)$, where $1 \leq deg^d(s_f(x)) \leq deg^d(s_g(x)) + deg^d(s_h(x))$ and $1 \leq deg^d(t_f(x)) \leq deg^d(t_g(x)) + deg^d(t_h(x))$.*
2. *Then $f(x) = \frac{s_f(x)}{t_f(x)} = g(x) + h(x)$, where $0 \leq deg^d(s_f(x)) \leq \max(deg^d(s_g(x)) + deg^d(t_h(x)), deg^d(s_h(x)) + deg^d(t_g(x)))$ and $0 \leq deg^d(t_f(x)) \leq deg^d(t_g(x)) + deg^d(t_h(x))$.*
3. *Then $f(x) = \frac{s_f(x)}{t_f(x)} = g'(x)$, where $deg^d(s_f(x)) \leq deg^d(s_g(x)) + deg^d(t_g(x))$ and $deg^d(t_f(x)) \leq 2deg^d(t_g(x))$.*
4. *Then $f(x) = \frac{s_f(x)}{t_f(x)} = g^q_m(x)$, where $deg^d(s_f(x)) \leq deg^d(s_g(x)) + (m-1)deg^d(t_g(x))$ and $deg^d(t_f(x)) \leq m \times deg^d(t_g(x)))$.*

Proof:

1. By letting $s_f(x) = s_g(x)s_h(x)$ and $t_f(x) := t_g(x)t_h(x)$ the upper bounds follows from Lemma 2.4. The lower bounds follow by taking $f(x) = \frac{1}{g(x)}$.
2. By letting $s_f(x) = s_g(x)t_h(x) + s_h(x)t_g(x)$ and $t_f(x) = t_g(x)t_h(x)$ the upper bounds follow from Lemma 2.4. The lower bounds follow by taking $f(x) = -g(x)$.
3. By letting $s_f(x) = s'_g(x)t_g(x) - s_g(x)t'_g(x)$ and $t_f(x) = t^2_g(x)$ the upper bounds follow from Lemma 2.4.
4. By letting $s_f(x) = (s_g(x)\prod_{i=1}^{m-1} t_g(x\omega^i_m))^q_m$ and $t_f(x) = (\prod_{i=0}^{m-1} t_g(x\omega^i_m))^0_m$ the upper bounds follow from Lemma 2.4.

■

Lemma 3.6 (*deg^P*.) *Let $h(x) = \frac{s_h(x)}{t_h(x)}$, $g(x) = \frac{s_g(x)}{t_g(x)} \in \mathcal{R}$, such that $s_h(x), t_h(x), s_g(x), t_g(x) \in \mathcal{P}$.*

1. *Then $f(x) = \frac{s_f(x)}{t_f(x)} = g(x)h(x)$, where $1 \leq deg^P(s_f(x)) \leq deg^P(s_g(x))deg^P(s_h(x))$ and $1 \leq deg^P(t_f(x)) \leq deg^P(t_g(x))deg^P(t_h(x))$.*

2. *Then* $f(x) = \frac{s_f(x)}{t_f(x)} = g(x) + h(x)$, *where* $0 \le deg^P(s_f(x)) \le deg^P(s_g(x))deg^P(t_h(x)) + deg^P(s_h(x))\ deg^P(t_g(x)))$ *and* $1 \le deg^P(t_f(x)) \le deg^P(t_g(x))deg^P(t_h(x))$.

3. *Then* $f(x) = \frac{s_f(x)}{t_f(x)} = g'(x)$, *where* $deg^P(s_f(x)) \le 2deg^P(s_g(x))deg^P(t_g(x))$ *and* $deg^P(t_f(x)) \le deg^P(t_g(x))^2$.

4. *Then* $f(x) = \frac{s_f(x)}{t_f(x)} = g^q_m(x)$, *where* $deg^P(s_f(x)) \le m \times deg^P(s_g(x))deg^P(t_g(x))^{m-1}$ *and also that* $deg^P(t_f(x)) \le deg^P(t_g(x))^m$.

Proof:

1. By letting $s_f(x) = s_g(x)s_h(x)$ and $t_f(x) = t_g(x)t_h(x)$ the upper bounds follows from Lemma 2.5. The lower bounds follow by taking $f(x) = \frac{1}{g(x)}$.

2. By letting $s_f(x) = s_g(x)t_h(x) + s_h(x)t_g(x)$ and $t_f(x) = t_g(x)t_h(x)$ the upper bounds follow from Lemma 2.5. The lower bounds follow by taking $f(x) = -g(x)$.

3. By letting $s_f(x) = s'_g(x)t_g(x) - s_g(x)t'_g(x)$ and $t_f(x) = t^2_g(x)$ the upper bounds follow from Lemma 2.5.

4. By letting $s_f(x) = (s_g(x)\prod_{i=1}^{m-1} t_g(x\omega^i_m))^q_m$ and $t_f(x) = (\prod_{i=0}^{m-1} t_g(x\omega^i_m))^0_m$ the upper bounds follow from Lemma 2.5.

$\blacksquare$

Note 3.2 *It is worth noting that the metrics under the operations of* $f(x) \to f(\alpha x)$ *was not examined as nothing interesting happens, and integration of functions in* $\mathcal{R}$ *was not examined as* $\mathcal{R}$ *is not closed under integration.*

These bounds will be used later in Chapter 5, as many methods to determine lacunary recurrence relations require bounds on the , size of these lacunary recurrence relations and also bounds on the multiplicity of the roots associated with their recurrence polynomials.

3.7 Examples.

This section does three detailed examples. That of $f(x) = \frac{1}{p(x)} \in \hat{\mathcal{R}}$ with $p(x)$ a polynomial, of $g(x) = \frac{1}{\sum_{i=1}^n \alpha_i e^{\lambda_i x}} \in \hat{\mathcal{R}}$, and lastly the Bernoulli polynomials.

Example 16 *Consider* $f(x) = \frac{1}{p(x)} \in \hat{\mathcal{R}}$. *Let* $p(x) = \alpha_n x^n + ... + \alpha_0$. *As* $f(x) \in \hat{\mathcal{R}}$, *notice that* $a_0 \neq 0$.

Then:

$$\begin{aligned}\sum_{k=0}^{\infty} c_k \frac{x^k}{k!} &= \frac{1}{\alpha_n x^n + ... + \alpha_0} \\ \sum_{i=0}^{n} \alpha_i i! \frac{x^i}{i!} \sum_{k=0}^{\infty} c_k \frac{x^k}{k!} &= 1 \\ \sum_{k=0}^{\infty} \sum_{i=0}^{n} \binom{k}{i} c_{k-i} \alpha_i i! \frac{x^k}{k!} &= 1.\end{aligned}$$

Considering $k = 0$ gives $c_0 = \frac{1}{\alpha_0}$, and considering $k > 0$ demonstrates that:

$$\begin{aligned}\sum_{i=0}^{n} \binom{k}{i} c_{k-i} \alpha_i i! &= 0 \\ c_k = \frac{-1}{\alpha_0} \sum_{i=1}^{n} \binom{k}{i} c_{k-i} \alpha_i i! &= 0.\end{aligned}$$

So a recursion formula for c_k was derived that only requires the previous $n-1$ terms.

Example 17 *Consider $g(x) \in \hat{\mathcal{R}}$ where $g(x) = \frac{1}{\sum_{i=1}^{n} \alpha_i e^{\lambda_i x}}$. A simple calculation gives $s(x) = \frac{1}{\sum_{i=0}^{\infty} b_i \frac{x^i}{i!}}$, where the $b_j = \sum_{i=1}^{n} \alpha_i \lambda_i^j$. This example will use this knowledge throughout.*

Hence:

$$\begin{aligned}\sum_{k=0}^{\infty} c_k \frac{x^k}{k!} &= \frac{1}{\sum_{j=0}^{\infty} \sum_{i=1}^{n} \alpha_i \lambda_i^j \frac{x^j}{j!}} \\ \sum_{j=0}^{\infty} \sum_{i=1}^{n} \alpha_i \lambda_i^j \frac{x^j}{j!} \sum_{k=0}^{\infty} c_k \frac{x^k}{k!} &= 1 \\ \sum_{j=0}^{\infty} \sum_{k=0}^{j} \binom{j}{k} c_k \sum_{i=1}^{n} \alpha_i \lambda_i^{j-k} \frac{x^j}{j!} &= 1.\end{aligned}$$

Considering $k = 0$ shows that $c_0 = \frac{1}{\sum_{i=1}^{n} a_i}$. As $g(x) \in \hat{\mathcal{R}}$ it follows that $c_0 \neq 0$. Considering $k > 0$ gives:

$$c_k = \frac{1}{\sum_{i=1}^{n} a_i} \left(- \sum_{j=1}^{k} \binom{k}{i} \sum_{i=1}^{n} \alpha_i \lambda_i^j c_{m-j}\right).$$

Example 18 *Consider the following example in Maple.*

```
>  with(MS):
```

This example will demonstrate how the methods of multisectioning can be applied to functions with symbolic parameters for parameters of the exponentials of rational poly-exponential functions. Define *the* "Bernoulli polynomials" *to be the coefficients of the exponential generating function of* $\frac{x\, e^{(t\, x)}}{e^x - 1}$ *in* x. *The denominator and numerator of this function have very complicated lacunary recurrence relations, even when multisectioning by a small value such as 3 (at 0).*

```
>   top := x* exp(t*x):
>   bot := exp(x)-1:
>   botlrr := `bottom/ms`(bot, f, x, 3);
```

$$\mathit{botlrr} := \mathrm{f}(x) = \mathrm{f}(x-6),\ f,\ x,\ [\mathrm{f}(0) = 0,\ \mathrm{f}(1) = 0,\ \mathrm{f}(2) = 0,\ \mathrm{f}(3) = 6,\ \mathrm{f}(4) = 0,\ \mathrm{f}(5) = 0]$$

```
>   toplrr := collect([`top/ms/linalg/sym`(top,bot, f, x, 3, 0)],f);
```

$$\begin{aligned}
\mathit{toplrr} := [&\mathrm{f}(x) = (-7152\,t^{14} - 7152\,t^{16} + 1932\,t^{11} - 3599\,t^{18} - 840\,t^{20} - t^6 + 5544\,t^{17}\\
&+ 7780\,t^{15} + 286\,t^{21} + 5544\,t^{13} + 12\,t^7 - 72\,t^{22} - t^{24} - 72\,t^8 + 1932\,t^{19}\\
&+ 12\,t^{23} - 840\,t^{10} + 286\,t^9 - 3599\,t^{12})\mathrm{f}(x-24) + 2(4\,t^{18} - 42\,t^{17} + 216\,t^{16}\\
&- 722\,t^{15} + 1764\,t^{14} - 3366\,t^{13} + 5244\,t^{12} - 6894\,t^{11} + 7836\,t^{10} - 7813\,t^9\\
&+ 6852\,t^8 - 5238\,t^7 + 3427\,t^6 - 1872\,t^5 + 828\,t^4 - 285\,t^3 + 72\,t^2 - 12\,t + 1)\\
&t^3\,\mathrm{f}(x-21) + (-28\,t^{18} + 252\,t^{17} - 1080\,t^{16} + 2928\,t^{15} - 5688\,t^{14} + 8568\,t^{13}\\
&- 10578\,t^{12} + 11052\,t^{11} - 9960\,t^{10} + 7978\,t^9 - 5976\,t^8 + 4320\,t^7 - 2910\,t^6\\
&+ 1692\,t^5 - 792\,t^4 + 282\,t^3 - 72\,t^2 + 12\,t - 1)\mathrm{f}(x-18) + (56\,t^{15} - 420\,t^{14}\\
&+ 1440\,t^{13} - 2990\,t^{12} + 4272\,t^{11} - 4620\,t^{10} + 4066\,t^9 - 2952\,t^8 + 1536\,t^7\\
&- 202\,t^6 - 552\,t^5 + 612\,t^4 - 346\,t^3 + 120\,t^2 - 24\,t + 2)\mathrm{f}(x-15) + (-70\,t^{12}\\
&+ 420\,t^{11} - 1080\,t^{10} + 1550\,t^9 - 1368\,t^8 + 792\,t^7 - 354\,t^6 + 180\,t^5 - 120\,t^4\\
&+ 74\,t^3 - 24\,t^2 + 1)\mathrm{f}(x-12) +\\
&(56\,t^9 - 252\,t^8 + 432\,t^7 - 336\,t^6 + 72\,t^5 + 72\,t^4 - 24\,t^3 - 36\,t^2 + 24\,t - 4)\\
&\mathrm{f}(x-9) + (-28\,t^6 + 84\,t^5 - 72\,t^4 + 4\,t^3 + 24\,t^2 - 12\,t + 1)\,\mathrm{f}(x-6)\\
&+ (8\,t^3 - 12\,t^2 + 2)\,\mathrm{f}(x-3),\ f,\ x, [\mathrm{f}(0) = 0,\ \mathrm{f}(1) = 0,\ \mathrm{f}(2) = 0,\ \mathrm{f}(3) = 6,\ \mathrm{f}(4) = 0,\\
&\mathrm{f}(5) = 0,\ \mathrm{f}(6) = 60\,t + 120\,t^3 - 180\,t^2,\ \mathrm{f}(7) = 0,\ \mathrm{f}(8) = 0,\\
&\mathrm{f}(9) = 18 - 252\,t^2 + 1260\,t^4 + 504\,t^6 - 1512\,t^5,\ \mathrm{f}(10) = 0,\ \mathrm{f}(11) = 0,\\
&\mathrm{f}(12) = 264\,t + 3960\,t^3 - 1980\,t^2 + 7920\,t^7 - 5940\,t^8 - 5544\,t^5 + 1320\,t^9,\\
&\mathrm{f}(13) = 0,\ \mathrm{f}(14) = 0, \mathrm{f}(15) = 30 - 1365\,t^2 + 30030\,t^4 - 16380\,t^{11} + 90090\,t^6\\
&- 45045\,t^8 + 30030\,t^{10} - 90090\,t^5 + 2730\,t^{12},\ \mathrm{f}(16) = 0,\ \mathrm{f}(17) = 0, \mathrm{f}(18) =\\
&612\,t - 36720\,t^{14} + 24480\,t^3 - 7344\,t^2 - 222768\,t^{11} + 4896\,t^{15} + 85680\,t^{13}\\
&+ 700128\,t^7 - 1312740\,t^8 - 111384\,t^5 + 875160\,t^9,\ \mathrm{f}(19) = 0,\ \mathrm{f}(20) = 0,\\
&\mathrm{f}(21) = 42 - 813960\,t^{14} - 3990\,t^2 + 203490\,t^4 + 203490\,t^{16} - 10581480\,t^{11}\\
&+ 7980\,t^{18} + 1627920\,t^6 - 71820\,t^{17} - 2645370\,t^8 + 7759752\,t^{10}\\
&- 976752\,t^5 + 5290740\,t^{12},\ \mathrm{f}(22) = 0,\ \mathrm{f}(23) = 0]]
\end{aligned}$$

Now, if $t = 0$ this will reduce to the situation of looking at the normal Bernoulli numbers.

```
>   subs(t=0,[toplrr]);
```

$$[[\mathrm{f}(x) = -\mathrm{f}(x-18) + 2\,\mathrm{f}(x-15) + \mathrm{f}(x-12) - 4\,\mathrm{f}(x-9) + \mathrm{f}(x-6) + 2\,\mathrm{f}(x-3),\ f,\ x,[$$
$$\mathrm{f}(0) = 0,\ \mathrm{f}(1) = 0,\ \mathrm{f}(2) = 0,\ \mathrm{f}(3) = 6,\ \mathrm{f}(4) = 0,\ \mathrm{f}(5) = 0,\ \mathrm{f}(6) = 0,\ \mathrm{f}(7) = 0,\ \mathrm{f}(8) = 0,$$
$$\mathrm{f}(9) = 18,\ \mathrm{f}(10) = 0,\ \mathrm{f}(11) = 0,\ \mathrm{f}(12) = 0,\ \mathrm{f}(13) = 0,\ \mathrm{f}(14) = 0,\ \mathrm{f}(15) = 30,$$
$$\mathrm{f}(16) = 0,\ \mathrm{f}(17) = 0,\ \mathrm{f}(18) = 0,\ \mathrm{f}(19) = 0,\ \mathrm{f}(20) = 0,\ \mathrm{f}(21) = 42,\ \mathrm{f}(22) = 0,$$
$$\mathrm{f}(23) = 0]]]$$

This example is interesting because it demonstrates how large and complicated the results get when done symbolically, but still shows that feasibility of doing these calculations.

3.8 Conclusion.

By combining Theorem 3.1, Lemmas 3.3, 3.4 and 3.6 the follow results follow: Although some corollaries of this result are know, (for examples, for the particular cases of the Bernoulli, Euler, Genocchi, or Lucas type II numbers), to the best of my knowledge, they have not been done to this degree of generality before

Theorem 3.3 *Let $f(x) \in \hat{\mathcal{R}}$, $m, q \in \mathbb{Z}$, $0 \le q < m$.*

1. *Then a lacunary recursion formula can be found for the $mi + q$-th coefficient of the exponential generating function of $f(x)$ that depends only on the $mj + q$-th coefficient, for $j = 0, 1, ..., i-1$, and two lacunary recurrence relations.*
2. *Moreover, if $f(x) = \frac{s(x)}{t(x)}$ then upper bounds on the length of the two lacunary recurrence relations are $m \times deg^P(s(x))deg^P(t(x))^{m-1}$ for the numerator and $deg^P(t(x))^m$ for the denominator.*
3. *Furthermore if $f(x) \in \hat{\mathcal{R}}^{R_1,R_2}$, then the two lacunary recurrence relations are both in $\mathcal{P}^{R_1\langle\omega_m^i\rangle,R_2}$.*
4. *Lastly, if the recurrence polynomials of $s(x)$ and $t(x)$ are in $R_3[x]$, then the recurrence polynomials of the two lacunary recurrence relations are in $R_3[x]$.*

Corollary 9 *A lacunary recursion formula can be found for the $(mi + q)$-th Bernoulli number that depends only on the $(mj+q)$-th Bernoulli number, for $j = 0, 1, ..., i-1i$, and two lacunary recurrence relations, with upper bounds on their sizes of $m2^m$ and 2^m respectively, where all the terms of the lacunary recurrence relations and the recurrence polynomials themselves are in $\mathbb{Z}$.*

Note 3.3 *Tighter upper bounds for the sizes of the lacunary recurrence relations were determined by Chellali [9] for the Bernoulli numbers. This was*

$$\sum_{d|m,\mathrm{odd}} \mu(d) 2^{m/d}/2m$$

for the lacunary recurrence relation that is derived from the denominators and twice this for that of the numerator, when multisectioning by m. Here μ is the Mobius function, as defined in [2]. This result requires specialized techniques and does not follow directly follow from any of the results in this thesis.

Chapter 4

Calculations of recurrences for $\mathcal{P}$.

In the previous chapters a very naive approach was used to calculate the lacunary recurrence relations that would be needed for the calculation of the coefficients to the exponential generating functions of the functions in both $\mathcal{P}$ and $\mathcal{R}$. The function's representation as polynomials and exponential functions, was naively multisectioned using the formula in Definition 2.6. After this, the multisectioned function was converted to a formula where the lacunary recurrence relation could be observed. The goal of the next two chapters is to show some other, more efficient ways, by which these lacunary recurrence relations and lacunary recursion formulae can be computed.

In this chapter, different methods to multisection functions in $\mathcal{P}$ are examined, and Chapter 5 examines different methods for those functions in $\mathcal{R}$.

Section 4.1 looks at how to use recurrence polynomials to multisection poly-exponential functions. This method takes advantage of the factorization of m, the quantity by which the poly-exponential function is multisectioned. Section 4.2 looks at how to use recurrence polynomials and resultants to multisection poly-exponential functions. Using linear algebra to find the new lacunary recurrence relations of a poly-exponential functions that are multisectioned, as well as how to use symbolic differentiation with linear algebra is looked at in Section 4.3 and 4.4. Section 4.5 looks at how to take advantage of the factorization of m, by iteratively compressing the results. Section 4.6 and 4.7 looks at two theories where by the problem being studied can be simplified. The last section, Section 4.8, makes some conclusions based on empirical evidence as to which methods are best.

4.1 Multisectioning the recurrence polynomial.

Recall that if $s(x) \in \mathcal{P}$ then $P^s(x)$ is the recurrence polynomial associated with $s(x)$ (Definition 2.2). The first thing needed was shown in Corollary 2 and Lemma 2.5 which is reiterated here:

Lemma 4.1 *If $s(x)$, $t(x) \in \mathcal{P}$, $\alpha \neq 0$ where $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$ and where $t(x) = \sum_{j=1}^{m} q_j(x)e^{\mu_j x}$ then:*

1. $P^{st}(x) | \prod_{i=1,j=1}^{i=n,j=m}(x - \lambda_i - \mu_j)^{\deg(p_i(x))+\deg(q_i(x))}$,
2. $P^{s+t}(x)|P^s(x)P^t(x)$,
3. $P^{s(\alpha x)}(x) = P^s(\alpha x)$,
4. $P^{\alpha s}(x) = P^s(x)$.

By using this information, the linear recurrence relation for a poly-exponential function may be multisectioned by only looking at the recurrence polynomial.

Lemma 4.2 *If $s(x) \in \mathcal{P}$ then*

$$P^{s^q_m(x)}(x) | \prod_{i=0}^{m-1} P^s(x\omega^i_m).$$

Proof: By noticing that $P^{s+t}(x)|P^s(x)P^t(x)$, and $P^{s(\alpha x)}(x) = P^s(\alpha x)$ from Lemma 4.1, it follows that:

$$P^{s^q_m(x)}(x) \quad = \quad P^{\frac{1}{m}\sum_{i=0}^{m-1}\omega_m^{-qi}s(x\omega^i_m)}(x) | \prod_{i=0}^{m-1} P^{s(x\omega^i_m)}(x) = \prod_{i=0}^{m-1} P^s(x\omega^i_m).$$

■

By recalling that any polynomial which the recurrence polynomial divides is a valid recurrence polynomial (Section 2.3), the above product $\prod_{i=0}^{m-1} P^s(x\omega^i_m)$ will give a valid lacunary recurrence relation for $s^q_m(x)$. Further it is fairly easy to do this computationally. With the additional information of $deg^d(s(x))$, an even better recurrence polynomial can be found, as $deg^d(s^q_m(x)) = deg^d(s(x))$ (Lemma 2.4). Hence this shows that the recurrence polynomial can have no roots of multiplicity greater than $deg^d(s(x)) + 1$ (Corollary 1).

From a computational point of view, the order in which the $P^s(x\omega^i_m)$ for $0 \leq i \leq m-1$ are multiplied together is important. For example if $m = 2^k$ and $P^s(x) \in \mathbb{Z}[x]$ then: $P^s(x), P^s(-x) \in \mathbb{Z}[x]$, and further that $P^s(x)P^s(-x) \in \mathbb{Z}[x^2]$. It follows that $P^s(ix)P^s(-ix) \in \mathbb{Z}[x^2]$ and hence $P^s(x)P^s(-x)P^s(ix)P^s(-ix) \in \mathbb{Z}[x^4]$. Etc.

In general, if $m = d_1 d_2 ... d_k$, for $d_i \in \mathbb{Z}$ where $2 \le d_i$, then this computation is best done as:

$$\prod_{i_k=0}^{d_k-1} \cdots \prod_{i_2=0}^{d_2-1} \prod_{i_1=0}^{d_1-1} P^s(x\omega_{d_1}^{i_1}\omega_{d_1 d_2}^{i_2} ... \omega_{d_1 d_2 .. d_k}^{i_k}),$$

performing the computation at the inner levels first, and using scaling to perform the next level out.

As a result of implementing this, a bug in Maple was found, which made the original method to scaling very inefficient. See Appendix D Section D.1 for more information about this.

Example 19 *Consider the following example in Maple. For more information about the Maple code, see Appendix A. For the Maple code see Appendix E. The Maple code and help files (including information about syntax) are available on the web at [1].*

```
>  with(MS):
```

Consider the exponential generating function $\mathrm{s}(x) = \sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}$ *with a linear recurrence relation* $b_i = b_{i-1} - b_{i-2} + b_{i-3}$*, with initial values of* $b_0 = 1$*,* $b_1 = 1$ *and* $b_2 = 1$*. This example multisections this linear recurrence relation by 16 at 0, using the methods described in this section. First determine the value of* $deg^d(s(x))$*.*

```
>  s := b(x) = b(x-1)-b(x-2)+b(x-3), b, x, [b(0) = 1, b(1) = 1, b(2) = 1];
```

$$\mathrm{s} := \mathrm{b}(x) = \mathrm{b}(x-1) - \mathrm{b}(x-2) + \mathrm{b}(x-3),\ b,\ x,\ [\mathrm{b}(0) = 1,\ \mathrm{b}(1) = 1,\ \mathrm{b}(2) = 1]$$

```
>  `egf/metric/d`(s);
```

$$0$$

From this it follows that the multisectioned recurrence polynomial can have no multiple roots.

So now determine the recurrence polynomial.

```
>  P := convert_poly(s);
```

$$P := x^3 - x^2 + x - 1$$

Now multiply $\mathrm{P}(x)$ *by* $\mathrm{P}(-x)$ *and expand.*

```
>  P2 := expand(subs(x=-x,P)*P);
```

$$P2 := -x^6 - x^4 + x^2 + 1$$

Now this polynomial should have no multiple roots, so get rid of the multiple roots.

```
>  P2p := quo(P2,gcd(P2, diff(P2,x)),x);
```

$$P2p := -x^4 + 1$$

Now multiply $\mathrm{P2p}(x)$ *by* $\mathrm{P2p}(x\,I)$*, and expand. This gives a recurrence polynomial that divides* $\mathrm{P}(x)\,\mathrm{P}(-x)\,\mathrm{P}(I\,x)\,\mathrm{P}(-I\,x)$ *and has no multiple roots.*

```
> P4 := expand(subs(x=x*I,P2p)*P2p);
```

$$P4 := x^8 - 2\,x^4 + 1$$

Again, get rid of the multiple roots.

```
> P4p := quo(P4, gcd(P4, diff(P4,x)),x);
```

$$P4p := x^4 - 1$$

Lastly, multiply $\mathrm{P4p}(x)$ *by* $\mathrm{P4p}(x\,\sqrt{I})$ *and expand. This gives a recurrence polynomial that divides* $\mathrm{P}(x)\,\mathrm{P}(-x)\,\mathrm{P}(I\,x)\,\mathrm{P}(-I\,x)\,\mathrm{P}(\sqrt{I}\,x)\,\mathrm{P}(-\sqrt{I}\,x)\,\mathrm{P}(I\,\sqrt{I}\,x)\,\mathrm{P}(-I\,\sqrt{I}\,x)$ *and has no multiple roots.*

```
> P8 := expand(subs(x=x*sqrt(I),P4p)*P4p);
```

$$P8 := -x^8 + 1$$

Again, get rid of the multiple roots.

```
> P8p := quo(P8, gcd(P8, diff(P8,x)),x);
```

$$P8p := -x^8 + 1$$

So converting back gives a linear recurrence relation of:

```
> convert_rec(P8p,b,x);
```

$$\mathrm{b}(x) = \mathrm{b}(x-8)$$

This is the same linear recurrence relation that is derived using the naive technique discussed in Example 13.

```
> `egf/ms/naive`(s,8,0);
```

$$\mathrm{b}(x) = \mathrm{b}(x-8),\ b,\ x,$$
$$[\mathrm{b}(0) = 1,\ \mathrm{b}(1) = 0,\ \mathrm{b}(2) = 0,\ \mathrm{b}(3) = 0,\ \mathrm{b}(4) = 0,\ \mathrm{b}(5) = 0,\ \mathrm{b}(6) = 0,\ \mathrm{b}(7) = 0]$$

This has been automated as the Maple command ‘egf/ms/rec‘.

```
> `egf/ms/rec`(s,8,0);
```

$$\mathrm{b}(x) = \mathrm{b}(x-8),\ b,\ x,$$
$$[\mathrm{b}(0) = 1,\ \mathrm{b}(1) = 0,\ \mathrm{b}(2) = 0,\ \mathrm{b}(3) = 0,\ \mathrm{b}(4) = 0,\ \mathrm{b}(5) = 0,\ \mathrm{b}(6) = 0,\ \mathrm{b}(7) = 0]$$

4.2 Multisectioning via resultants.

In the previous section, the recurrence polynomials of $s(x) \in \mathcal{P}$, say $P^s(x)$, was multisectioned by computing $\prod_{i=0}^{m-1} P^s(x\omega_m^i)$ in a naive fashion, and then getting rid of root with too high of an order. This section again computes $\prod_{i=0}^{m-1} P^s(x\omega_m^i)$ but in a more sophisticated manner; by using resultants [20].

Definition 4.1 *Let $p(x) = a\prod_{i=1}^{n}(x-\lambda_i)$ and $q(x) = b\prod_{j=1}^{m}(x-\mu_j)$. The "resultant", denoted $\operatorname{Res}_x(p(x), q(x))$ is defined as:*

$$\operatorname{Res}_x(p(x), q(x)) = a^m b^n \prod_{i=1,j=1}^{i=n,j=m} (\lambda_i - \mu_j).$$

This next theorem follows from the definition of the resultant.

Theorem 4.1 *Let $s(x) \in \mathcal{P}$, and $P^s(x)$ be the recurrence polynomial for $s(x)$ and $P^{s_m^q(x)}(x)$ the recurrence polynomial for $s_m^q(x)$. Then:*

$$P^{s_m^q(x)}(x) | \operatorname{Res}_y(y^m - x^m, P^s(y))$$

Proof: Write $P^s(y) = \prod_{i=1}^{n}(y-\lambda_i)$. Notice that $y^m - x^m = \prod_{i=1}^{m}(y - \omega_m^i x)$. Thus from Lemma 4.2 it follows that $P^{s_m^q(x)}(x) | \prod_{j=0}^{m-1} P^s(x\omega_m^j)$. Further:

$$\prod_{j=0}^{m-1} P^s(x\omega_m^j) = \prod_{j=0}^{m-1}\prod_{i=1}^{n}(\omega_m^j x - \lambda_i) = \operatorname{Res}_y(y^m - x^m, P^s(y)).$$

Which is the desired result.

■

There are many good methods for computing resultants efficiently, in a symbolic setting. See, for example [12, 13].

Example 20 *Consider the following example in Maple.*

```
> with(MS):
```

Consider the example of the Padovan numbers defined in [28] . Let $\mathrm{s}(x) = \sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}$, *where* $b_i = b_{i-2} + b_{i-3}$ *and* $b_0 = 1$, $b_1 = 0$, *and* $b_2 = 1$. *Consider multisectioning this by 17 at 0. This example will do this by computing the resultant of* $\mathrm{P^s}(y)$ *with* $y^{17} - x^{17}$.

```
> s := b(y) = b(y-2) + b(y-3), b, y, [b(0) = 1, b(1) = 0, b(2) = 1];
```

$$\mathrm{s} := \mathrm{b}(y) = \mathrm{b}(y-2) + \mathrm{b}(y-3),\ b,\ y,\ [\mathrm{b}(0) = 1,\ \mathrm{b}(1) = 0,\ \mathrm{b}(2) = 1]$$

```
> poly := convert_poly(s);
```

$$poly := y^3 - y - 1$$

```
> poly := resultant(y^17-x^17,poly,y);
```

$$poly := -18\,x^{17} - 1 - 119\,x^{34} + x^{51}$$

```
> convert_rec(poly, f, x);
```

$$\mathrm{f}(x) = 18\,\mathrm{f}(x-34) + \mathrm{f}(x-51) + 119\,\mathrm{f}(x-17)$$

There is a command in Maple to do this called ‘egf/ms/result‘.

```
> ‘egf/ms/result‘(s,17,0);
```

$$\begin{aligned}&\mathrm{b}(y) = 18\,\mathrm{b}(y-34) + \mathrm{b}(y-51) + 119\,\mathrm{b}(y-17),\ b,\ y, [\mathrm{b}(0) = 1,\ \mathrm{b}(1) = 0,\ \mathrm{b}(2) = 0,\\ &\quad \mathrm{b}(3) = 0,\ \mathrm{b}(4) = 0,\ \mathrm{b}(5) = 0,\ \mathrm{b}(6) = 0,\ \mathrm{b}(7) = 0,\ \mathrm{b}(8) = 0,\ \mathrm{b}(9) = 0,\ \mathrm{b}(10) = 0,\\ &\quad \mathrm{b}(11) = 0,\ \mathrm{b}(12) = 0,\ \mathrm{b}(13) = 0,\ \mathrm{b}(14) = 0,\ \mathrm{b}(15) = 0,\ \mathrm{b}(16) = 0,\ \mathrm{b}(17) = 49,\\ &\quad \mathrm{b}(18) = 0,\ \mathrm{b}(19) = 0,\ \mathrm{b}(20) = 0,\ \mathrm{b}(21) = 0,\ \mathrm{b}(22) = 0,\ \mathrm{b}(23) = 0,\ \mathrm{b}(24) = 0,\\ &\quad \mathrm{b}(25) = 0,\ \mathrm{b}(26) = 0,\ \mathrm{b}(27) = 0,\ \mathrm{b}(28) = 0,\ \mathrm{b}(29) = 0,\ \mathrm{b}(30) = 0,\ \mathrm{b}(31) = 0,\\ &\quad \mathrm{b}(32) = 0,\ \mathrm{b}(33) = 0,\ \mathrm{b}(34) = 5842,\ \mathrm{b}(35) = 0,\ \mathrm{b}(36) = 0,\ \mathrm{b}(37) = 0,\ \mathrm{b}(38) = 0,\\ &\quad \mathrm{b}(39) = 0,\ \mathrm{b}(40) = 0,\ \mathrm{b}(41) = 0,\ \mathrm{b}(42) = 0,\ \mathrm{b}(43) = 0,\ \mathrm{b}(44) = 0,\ \mathrm{b}(45) = 0,\\ &\quad \mathrm{b}(46) = 0,\ \mathrm{b}(47) = 0,\ \mathrm{b}(48) = 0,\ \mathrm{b}(49) = 0,\ \mathrm{b}(50) = 0]\end{aligned}$$

This gives the same result.

4.3 Using linear algebra on $\mathcal{P}$.

If $s(x) \in \mathcal{P}$, and an upper bound on the size of the linear recurrence relation is known, then this linear recurrence relation can be determined by the early cases.

This can be written concisely as:

Lemma 4.3 *If $s(x) \in \mathcal{P}$ and $deg^P(s(x)) \leq N$ and $deg^d(s(x)) = k$, then $P^s(x)$ can be calculated by the first $2N + k$ values.*

This result is fairly well know, and can be found in a number of difference linear algebra text books as an application of linear algebra. It is included here for completeness sake.

Proof: If b_{k+1}, b_{k+2}, ..., b_{k+2N} are the initial values of some linear recurrence relation, then this leads to the following system of N linear equations:

$$\begin{aligned} a_N b_{k+1} + a_{N-1} b_{k+2} + ... + a_1 b_{k+N} &= b_{k+N+1} \\ a_N b_{k+2} + a_{N-1} b_{k+3} + ... + a_1 b_{k+N+1} &= b_{k+N+2} \\ &\vdots \\ a_N b_{k+N} + a_{N-1} b_{k+N+1} + ... + a_1 b_{k+2N-1} &= b_{k+2N}. \end{aligned}$$

There are N linear equations, and N unknowns $(a_1, ..., a_N)$, hence a solution exists. To rewrite this in the language of linear algebra, find the values $a_1, ..., a_N$ so that they satisfy the equation:

$$\begin{bmatrix} b_{k+1} & b_{k+2} & \cdots & b_{k+N} \\ b_{k+2} & b_{k+3} & \cdots & b_{k+N+1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k+N} & b_{k+N+1} & \cdots & b_{k+2N-1} \end{bmatrix} \begin{bmatrix} a_N \\ a_{N-1} \\ \vdots \\ a_1 \end{bmatrix} = \begin{bmatrix} b_{k+N+1} \\ b_{k+N+2} \\ \vdots \\ b_{k+2N} \end{bmatrix}.$$

■

If when solving for the a_1, ..., a_N above, a unique solution is not found, set a_N to zero, and see if that gives a unique solution. If not, set a_{N-1} to 0, and see if that gives a unique solution. Continue in this manner. In this way when a unique solution is found, it will be of the shortest possible length.

It is also worth noting that if the order of all the columns is reversed then the resulting matrix is a Toeplitz matrix (this would mean that the expected solution is also reversed). This is nice, because there is an $\mathcal{O}(n^2)$ algorithm for solving $n \times n$ Toeplitz matrix [15].

This algorithm was not implemented with the Maple package included with this thesis, as most of the problems would still finish in a reasonable amount of time with Maple's less efficient linear algebra package.

This lemma is of great use for the computation of Bernoulli numbers, as an upper bound for $\prod_{i=0}^{m-1}(e^{\omega_m^i x} - 1)$ is determined in a paper by Chellali [9], as being:

$$\sum_{d|m,\text{odd}} \mu(d) 2^{m/d}/2m. \tag{4.1}$$

Here μ is the Mobius function, as defined in [2]. Later in Section 5.2 of Chapter 5, it will be seen how to use this.

Example 21 *Consider the following example in Maple.*

```
> with(MS):
```

Consider the example of the Fibonacci numbers. Let $\mathrm{s}(x) = \sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}$, *where* $b_0 = 0$ *and* $b_1 = 1$. *Consider multisectioning this by 17 at 0. From Lemma 2.5, the size of the new linear recurrence relation will be at most 17 times* $deg^P(s(x)) = 2$. *Further* $deg^d(s(x)) = 0$ *so it follows that the values* b_1, b_2, ... $b_{17\times2\times2}$ *are needed. All but* b_{17}, b_{34}, b_{51}, *and* b_{68} *will be zero, so only these four values are needed to determine the linear recurrence relation.*

```
> s := b(i) = b(i-1) + b(i-2), b, i, [b(0) = 0, b(1) = 1];
```

$$\mathrm{s} := \mathrm{b}(i) = \mathrm{b}(i-1) + \mathrm{b}(i-2),\ b,\ i,\ [\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 1]$$

```
> `egf/metric/P`(s);
```

$$2$$

```
> `egf/metric/d`(s);
```

$$0$$

```
> Fib := `egf/makeproc`(s):
```

So this gives the following two linear equations:

```
> eqn1 := a[1] * Fib(17) + a[2] * Fib(34) = Fib(51);
```

$$eqn1 := 1597\, a_1 + 5702887\, a_2 = 20365011074$$

```
> eqn2 := a[1] * Fib(34) + a[2] * Fib(51) = Fib(68);
```

$$eqn2 := 5702887\, a_1 + 20365011074\, a_2 = 72723460248141$$

Solving these two equations gives a_1 *and* a_2.

```
> solve({eqn1, eqn2});
```

$$\{a_1 = 1,\ a_2 = 3571\}$$

So this gives the linear recurrence relation $b_i = 3571\, b_{i-17} + b_{i-28}$. *This could have also been solved by using the linear algebra package in Maple in the following way.*

```
> C = matrix(2,2,[Fib(17), Fib(34), Fib(34), Fib(51)]);
```

$$C := \begin{bmatrix} 1597 & 5702887 \\ 5702887 & 20365011074 \end{bmatrix}$$

```
> B = vector(2, [Fib(51), Fib(68)]);
```

$$B := [20365011074,\ 72723460248141]$$

```
> linsolve(C,B);
```

$$[1, 3571]$$

There is also a command in Maple to do this called ‘egf/ms/linalg‘.

```
> ‘egf/ms/linalg‘(s,17,0);
```

$$\mathrm{b}(i) = \mathrm{b}(i-34) + 3571\,\mathrm{b}(i-17),\ b,\ i, [\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 1,\ \mathrm{b}(2) = 0,\ \mathrm{b}(3) = 0,\ \mathrm{b}(4) = 0,\ \mathrm{b}(5) = 0,\ \mathrm{b}(6) = 0,\ \mathrm{b}(7) = 0,\ \mathrm{b}(8) = 0,\ \mathrm{b}(9) = 0,\ \mathrm{b}(10) = 0,\ \mathrm{b}(11) = 0,\ \mathrm{b}(12) = 0,\ \mathrm{b}(13) = 0,\ \mathrm{b}(14) = 0,\ \mathrm{b}(15) = 0,\ \mathrm{b}(16) = 0,\ \mathrm{b}(17) = 0,\ \mathrm{b}(18) = 2584,\ \mathrm{b}(19) = 0,\ \mathrm{b}(20) = 0,\ \mathrm{b}(21) = 0,\ \mathrm{b}(22) = 0,\ \mathrm{b}(23) = 0,\ \mathrm{b}(24) = 0,\ \mathrm{b}(25) = 0,\ \mathrm{b}(26) = 0,\ \mathrm{b}(27) = 0,\ \mathrm{b}(28) = 0,\ \mathrm{b}(29) = 0,\ \mathrm{b}(30) = 0,\ \mathrm{b}(31) = 0,\ \mathrm{b}(32) = 0,\ \mathrm{b}(33) = 0]$$

So this again gives the same result.

4.4 Using symbolic differentiation with linear algebra.

Section 4.3 used knowledge about what the linear recurrence relation to determine the first $2N + k$ cases, (N and k defined as before). If $s(x)$ is function instead in poly-exponential form, then symbolic differentiation can be used to find the first $2N + k$ cases.

Example 22 *Consider the following example in Maple.*

```
> with(MS):with(linalg):
```

Consider the poly-exponential function $\mathrm{s}(x) = e^{(2\,x)}\,x^3 + e^{(3\,x)}$*. Notice that* $deg^P(s(x)) = 5$ *and* $deg^d(s(x)) = 3$*. Hence to multisection by 7 at 4, we need only look at the values for* b_4*,* b_{11}*,* b_{18}*...,* b_{74}*.*

```
> s := exp(2*x)*x^3 + exp(3*x);
```

$$\mathrm{s} := e^{(2\,x)}\,x^3 + e^{(3\,x)}$$

```
> ‘pe/metric/P‘(s,x);
```

$$5$$

```
> ‘pe/metric/d‘(s,x);
```

$$3$$

```
> for i from 4 to 74 by 7 do
> b[i] := eval(diff(s,x£i),x=0);
> od;
```

$$b_4 := 129$$

$$b_{11} := 430587$$

$$b_{18} := 547852617$$

$$b_{25} := 905170004643$$

$$b_{32} := 1868997467192961$$

$$b_{39} := 4056323316806318091$$

$$b_{46} := 8863739267804963800569$$

$$b_{53} := 19383403919667326068655667$$

$$b_{60} := 42391187864946619249022072241$$

$$b_{67} := 92709468450045486192098346397467$$

$$b_{74} := 202755596822820624363186974870842281$$

Set the matrix C equal to $\begin{bmatrix} b_{11} & b_{18} & b_{25} & b_{32} & b_{39} \\ b_{18} & b_{25} & b_{32} & b_{39} & b_{46} \\ b_{25} & b_{32} & b_{39} & b_{46} & b_{53} \\ b_{32} & b_{39} & b_{46} & b_{53} & b_{60} \\ b_{39} & b_{46} & b_{53} & b_{60} & b_{67} \end{bmatrix}$.

```
> C := matrix(5,5,[seq(seq(b[4+7*(i+j-1)],i=1..5),j=1..5)]):
```

Set the vector v equal to $[b_{44}, b_{51}, b_{58}, b_{65}, b_{72}]$.

```
> v := vector(5, [seq(b[4+7*i+35],i=1..5)]):
```

Now solve.

```
> linsolve(C,v);
```

$$[587068342272,\ -18614321152,\ 223379456,\ -1218048,\ 2699]$$

This gives a linear recurrence relation of $d_i = 587068342272 d_{i-35} - 18614321152 b_{i-28} + 223379456$ $b_{i-21} - 1218048 b_{i-14} + 2699 b_{i-7}$.

This could have also been done by the Maple function 'pe/ms/linalg/sym'.

```
>   'pe/ms/linalg/sym'(s,f, x,7,2);
```

$$\begin{aligned}&\mathrm{f}(x) = 587068342272\,\mathrm{f}(x-35) - 18614321152\,\mathrm{f}(x-28) + 223379456\,\mathrm{f}(x-21)\\ &\quad - 1218048\,\mathrm{f}(x-14) + 2699\,\mathrm{f}(x-7),\ f,\ x, [\mathrm{f}(0)=0,\ \mathrm{f}(1)=0,\ \mathrm{f}(2)=9,\ \mathrm{f}(3)=0,\\ &\quad \mathrm{f}(4)=0,\ \mathrm{f}(5)=0,\ \mathrm{f}(6)=0,\ \mathrm{f}(7)=0,\ \mathrm{f}(8)=0,\ \mathrm{f}(9)=51939,\ \mathrm{f}(10)=0,\\ &\quad \mathrm{f}(11)=0,\ \mathrm{f}(12)=0,\ \mathrm{f}(13)=0,\ \mathrm{f}(14)=0,\ \mathrm{f}(15)=0,\ \mathrm{f}(16)=70571841,\\ &\quad \mathrm{f}(17)=0,\ \mathrm{f}(18)=0,\ \mathrm{f}(19)=0,\ \mathrm{f}(20)=0,\ \mathrm{f}(21)=0,\ \mathrm{f}(22)=0,\\ &\quad \mathrm{f}(23)=105285347403,\ \mathrm{f}(24)=0,\ \mathrm{f}(25)=0,\ \mathrm{f}(26)=0,\ \mathrm{f}(27)=0,\ \mathrm{f}(28)=0,\\ &\quad \mathrm{f}(29)=0,\ \mathrm{f}(30)=209160675948729,\ \mathrm{f}(31)=0,\ \mathrm{f}(32)=0,\ \mathrm{f}(33)=0,\\ &\quad \mathrm{f}(34)=0]\end{aligned}$$

Which is the same result.

4.5 Using compression.

In most situations, the main interest is the lacunary recurrence relations not the poly-exponential functions themselves. Define a new operation that will maintain the useful information of a lacunary recurrence relation such that the function under this operation will have a smaller recurrence polynomial.

Definition 4.2 **(C_m^q.)** *Define* C_m^q *that acts on* $\sum_{i=0}^{\infty} b_{mi+q} \frac{x^{mi+q}}{(mi+q)!}$ *by* $C_m^q(\sum_{i=0}^{\infty} b_{mi+q} \frac{x^{mi+q}}{(mi+q)!}) = \sum_{i=0}^{\infty} b_{im+q} \frac{x^i}{i!}$.

The term "*compressing*" will be used to describe this process. When saying a function $s(x)$ is "*compressed by m*", $C_m^q(s(x))$ is being looked at for some q. When saying a function $s(x)$ is "*compressed by m at q*", then $C_m^q(s(x))$ is being studied.

Methods similiar to those that arrive via compressing can be found for Fibonacci or Lucas numbers [16]. To the best of my knowledge, the definition, or consequences of compressing have not been written in this way before.

Some properties of compression are enumerated below.

Lemma 4.4 *Let* $s(x) \in \mathcal{P}$ *and let* R_1, R_2 *be subrings of* $\mathbb{C}$, *then:*

1. *If* $s^q_m(x) \in \mathcal{P}^{R_1,R_2}$ *then* $C^q_m(s^q_m(x)) \in \mathcal{P}^{R_1,R_2}$.
2. *If* $s^q_m(x) \in \mathcal{P}_{R_1,R_2}$ *then* $C^q_m(s^q_m(x)) \in \mathcal{P}_{R_1,R_2\langle R_1,R_1^{-1}\rangle}$.
3. *If* $P^{s^q_m(x)}(x) \in R_1[x]$ *then* $P^{C^q_m(s^q_m(x))}(x) \in R_1[x]$.
4. *Then* $deg^d(s^q_m(x)) \geq deg^d(C^q_m(s^q_m(x)))$.
5. *Then* $deg^P(s^q_m(x)) = m \times deg^P(C^q_m(s^q_m(x)))$.

Proof:

1. If $P^{s^q_m(x)}(x) = \prod_{i=1}^n (x^m - \lambda_i)$ then $P^{C^q_m(s^q_m(x)}(x) = \prod_{i=1}^n (x - \lambda_i)$, hence the recurrence polynomial for $C^q_m(s^q_m(x))$ splits in R_1. The coefficients of the exponential generating function are still in R_2, as they haven't changed value, only positions within the exponential generating function.

2. This follows from the hierarchy theorem (Theorem 2.2) as if $s^q_m(x) \in \mathcal{P}_{R_1,R_2}$ then $s^q_m(x) \in \mathcal{P}^{R_1,\langle R_1R_2,R_2\rangle}$. Hence from part 1 of this lemma, as $(s^q_m(x)) \in \mathcal{P}^{R_1,\langle R_1R_2,R_2\rangle}$ then $C^q_m(s^q_m(x)) \in \mathcal{P}^{R_1,\langle R_1R_2,R_2\rangle}$. This again from Theorem 2.2 gives that $C^q_m(s^q_m(x)) \in \mathcal{P}_{R_1,\langle R_1R_2,R_2\rangle\langle R_1,R_1^{-1}\rangle}$ which is equal to $\mathcal{P}_{R_1,R_2\langle R_1,R_1^{-1}\rangle}$.

3. If $P^{s^q_m(x)}(x) = x^{mn} + a_{n-1}x^{m(n-1)} + ...a_0$, then $P^{C^q_m(s^q_m(x))} = x^n + a_{n-1}x^{n-1} + ...a_0$. From this coefficients of $P^{C^q_m(s^q_m(x))}$ are still in R_1.

4. The recurrence polynomial of $s^q_m(x)$ can be written as a polynomial in x^m, say $\prod_{i=1}^n (x^m - \lambda_i)$. After the compression, the recurrence polynomial will be written as a polynomial in x, namely $\prod_{i=1}^n (x - \lambda_i)$. If some λ_i has multiplicity $deg^d(C^q_m(s^q_m(x)))$ in $\prod_{i=1}^n (x - \lambda_i)$, then λ_i will also appear with that multiplicity in $\prod_{i=1}^n (x^m - \lambda_i)$. From this $deg^d(s^q_m(x)) \geq deg^d(C^q_m(s^q_m(x)))$.

5. The recurrence polynomial of $s^q_m(x)$ can be written as a polynomial in x^m say $x^{mn} + a_{n-1}x^{m(n-1)}$ $+ ... + a_0$. After the compression, it will be written as a polynomial in x, namely $x^n + a_{n-1}x^{n-1} + ...a_0$, in x^m. This is a polynomial with the same coefficients, but with $\frac{1}{m}$-th the degree. Thus $deg^P(s^q_m(x)) = m \times deg^P(C^q_m(s^q_m(x)))$.

■

Theorem 4.2 *Let* $s(x) \in \mathcal{P}$, *with* $m = d_1...d_n$, *and* $q = a_1(d_2...d_n) + a_2(d_3...d_n) + ... + a_n$ *where* $0 \leq a_i < d_i$. *Consequently:*

$$C^q_m(s^q_m(x)) = C^{a_1}_{d_1}((C^{a_2}_{d_2}((...C^{a_n}_{d_n}(s^{a_n}_{d_n}(x)))^{a_{n-1}}_{d_{n-1}}...)^{a_1}_{d_1}).$$

Proof: Show that if $m = d_1 d_2$ and $q = a_2 d_1 + a_1$ for $d_i \in \mathbb{Z}$ where $2 \le d_i$, and $0 \le a_i < d_i$ then:

$$C_m^q(s_m^q(x)) = C_{d_1}^{a_1}((C_{d_2}^{a_2}(s_{d_2}^{a_2}(x)))_{d_1}^{a_1}).$$

and then the result will follow by induction.

Assume that $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$. Then:

$$\begin{aligned} C_{d_1}^{a_1}((C_{d_2}^{a_2}(s_{d_2}^{a_2}(x)))_{d_1}^{a_1}) &= C_{d_1}^{a_1}((C_{d_2}^{a_2}((\sum_{i=0}^{\infty} b_i \frac{x^i}{i!})_{d_2}^{a_2}))_{d_1}^{a_1}) = C_{d_1}^{a_1}((C_{d_2}^{a_2}(\sum_{i=0}^{\infty} b_{d_2 i+a_2} \frac{x^{d_2 i+a_2}}{(d_2 i+a_1)!}))_{d_1}^{a_1}) \\ &= C_{d_1}^{a_1}((\sum_{i=0}^{\infty} b_{d_2 i+a_2} \frac{x^i}{i!})_{d_1}^{a_1}) = C_{d_1}^{a_1}(\sum_{i=0}^{\infty} b_{d_1(d_2 i+a_2)+a_1} \frac{x^{d_1 i+a_1}}{(d_1 i+a_1)!}) \\ &= \sum_{i=0}^{\infty} b_{d_1(d_2 i+a_2)+a_1} \frac{x^i}{i!} = \sum_{i=0}^{\infty} b_{d_1 d_2 i+d_1 a_2+a_1} \frac{x^i}{i!} = \sum_{i=0}^{\infty} b_{mi+q} \frac{x^i}{i!}. \end{aligned}$$

But this is precisely $C_m^q(s_m^q(x))$, hence the result follows by induction.

∎

This is of great value as $C_{d_1}^{a_1}((C_{d_2}^{a_2}((...C_{d_n}^{a_n}(s_{d_n}^{a_n}(x)))_{d_{n-1}}^{a_{n-1}}...)_{d_1}^{a_1})$ is much easier to compute than is $C_m^q(s_m^q(x))$. This method of iteratively multisectioning requires less memory and time than doing the multisectioning process all in one calculation.

To see this, first let $f(m)$ be the complexity of the underlying algorithm that a poly-exponential function $s(x)$ is being multisectioned, when multisectioned by m. (This is something roughly linear for a fixed $s(x)$ but the exact order is not relevant to this argument.) Consider multisectioning by $m = p_1 p_2 ....p_n$, where p_i is a non-decreasing sequence of primes (not necessarily distinct). Then to iteratively perform this multisectioning by m requires $\mathcal{O}(f(p_1) + f(p_2) + ...f(p_n)) \le \mathcal{O}(mf(p_n))$. Thus even if $f(n) \ge n$ (i.e. $f(n)$ is worse than linear), and to multisection by a power of a prime p, say $m = p^n$, then the running time is logarithmic in m (regardless of the running time of the actual algorithm). (This ignores some of the problems associated with large integers, but is essentially correct.)

Example 23 *Consider the following example in Maple.*

```
>  with(MS):
```

This example looks at the Lucas numbers type I. Consider the linear recurrence relation $b_i = b_{i-1} + b_{i-2}$ where $b_0 = 2$ and $b_1 = 1$. Multisection this by 8 at 2. Notice that $8 = 2^3$ and further that 2 = 0 (4) + 1 (2) + 0. Any method can be used to compute the intermediate multisectioning. For this example the naive method is used.

So the first step is to calculate $s_2^0(x)$, where $\mathrm{s}(x) = \sum_{i=0}^{\infty} \frac{b_i x^i}{i!}$ with the b_is defined as above.

```
> s := b(i) = b(i-1) + b(i-2) , b, i, [b(0) = 2, b(1) = 1];
```

$$s := \mathrm{b}(i) = \mathrm{b}(i-1) + \mathrm{b}(i-2),\, b,\, i,\, [\mathrm{b}(0) = 2,\, \mathrm{b}(1) = 1]$$

```
> t := `egf/ms/naive`(s,2,0);
```

$$t := \mathrm{b}(i) = 3\,\mathrm{b}(i-2) - \mathrm{b}(i-4),\, b,\, i,\, [\mathrm{b}(0) = 2,\, \mathrm{b}(1) = 0,\, \mathrm{b}(2) = 3,\, \mathrm{b}(3) = 0]$$

Now compress this result.

```
> s2 := readlib(`egf/compress`)(t, 2, 0);
```

$$s2 := \mathrm{b}(i) = 3\,\mathrm{b}(i-1) - \mathrm{b}(i-2),\, b,\, i,\, [\mathrm{b}(0) = 2,\, \mathrm{b}(1) = 3]$$

The second step is to calculate the multisectioning of the above function s2 by 2 at 1.

```
> t2 := `egf/ms/naive`(s2, 2, 1);
```

$$t2 := \mathrm{b}(i) = 7\,\mathrm{b}(i-2) - \mathrm{b}(i-4),\, b,\, i,\, [\mathrm{b}(0) = 0,\, \mathrm{b}(1) = 3,\, \mathrm{b}(2) = 0,\, \mathrm{b}(3) = 18]$$

Now compress the result.

```
> s3 := `egf/compress`(t2, 2, 1);
```

$$s3 := \mathrm{b}(i) = 7\,\mathrm{b}(i-1) - \mathrm{b}(i-2),\, b,\, i,\, [\mathrm{b}(0) = 3,\, \mathrm{b}(1) = 18]$$

Now the last step is to multisection the above function s3 by 2 at 0.

```
> t3 := `egf/ms/naive`(s3, 2, 0);
```

$$t3 := \mathrm{b}(i) = 47\,\mathrm{b}(i-2) - \mathrm{b}(i-4),\, b,\, i,\, [\mathrm{b}(0) = 3,\, \mathrm{b}(1) = 0,\, \mathrm{b}(2) = 123,\, \mathrm{b}(3) = 0]$$

By compressing this result, a linear recurrence relation for the Lucas numbers type I is found using only every 8-th term.

```
> s4 := `egf/compress`(t3, 2, 0);
```

$$s4 := \mathrm{b}(i) = 47\,\mathrm{b}(i-1) - \mathrm{b}(i-2),\, b,\, i,\, [\mathrm{b}(0) = 3,\, \mathrm{b}(1) = 123]$$

Uncompress this result to get the answer, as expected from the other commands.

```
> readlib(`egf/uncompress`)(s4, 8, 2);
```

$$\mathrm{b}(i) = 47\,\mathrm{b}(i-8) - \mathrm{b}(i-16),\, b,\, i, [\mathrm{b}(0) = 0,\, \mathrm{b}(1) = 0,\, \mathrm{b}(2) = 3,\, \mathrm{b}(3) = 0,\, \mathrm{b}(4) = 0,\, \mathrm{b}(5) = 0,\, \mathrm{b}(6) = 0,\, \mathrm{b}(7) = 0,\, \mathrm{b}(8) = 0,\, \mathrm{b}(9) = 0,\, \mathrm{b}(10) = 123,\, \mathrm{b}(11) = 0,\, \mathrm{b}(12) = 0,\, \mathrm{b}(13) = 0,\, \mathrm{b}(14) = 0,\, \mathrm{b}(15) = 0]$$

Notice that using the naive method directly to multisection by 8 at 2 gives the same result, but the method takes much longer to work.

```
>   `egf/ms/naive`(s,8,2);
```

$$\mathrm{b}(i) = 47\,\mathrm{b}(i-8) - \mathrm{b}(i-16),\ b,\ i, [\mathrm{b}(0)=0,\ \mathrm{b}(1)=0,\ \mathrm{b}(2)=3,\ \mathrm{b}(3)=0,\ \mathrm{b}(4)=0,\ \mathrm{b}(5)=0,\ \mathrm{b}(6)=0,\ \mathrm{b}(7)=0,\ \mathrm{b}(8)=0,\ \mathrm{b}(9)=0,\ \mathrm{b}(10)=123,\ \mathrm{b}(11)=0,\ \mathrm{b}(12)=0,\ \mathrm{b}(13)=0,\ \mathrm{b}(14)=0,\ \mathrm{b}(15)=0]$$

This process has been automated with the Maple command 'egf/ms/compress'. The last option of the command specifies to use the naive method to do the underlying computation.

```
>   `egf/ms/compress`(s, 8, 2, naive);
```

$$\mathrm{b}(i) = 47\,\mathrm{b}(i-8) - \mathrm{b}(i-16),\ b,\ i, [\mathrm{b}(0)=0,\ \mathrm{b}(1)=0,\ \mathrm{b}(2)=3,\ \mathrm{b}(3)=0,\ \mathrm{b}(4)=0,\ \mathrm{b}(5)=0,\ \mathrm{b}(6)=0,\ \mathrm{b}(7)=0,\ \mathrm{b}(8)=0,\ \mathrm{b}(9)=0,\ \mathrm{b}(10)=123,\ \mathrm{b}(11)=0,\ \mathrm{b}(12)=0,\ \mathrm{b}(13)=0,\ \mathrm{b}(14)=0,\ \mathrm{b}(15)=0]$$

Which gives the same results.

4.6 Computing over the integers.

Doing calculations over the rationals is always expensive. This is because of the inherent problem of rational numbers of computing the greatest common divisor with every addition or multiplication. As well, memory requirements double for each addition of comparable sized rationals. For a more detailed description of these problems see Graham, Knuth and Patashnik's book *Concrete Mathematics* [16].

For this reason, it is desirable to perform the calculations over the integers if possible. Below are some conditions and techniques to get the computations to work for the integers.

Lemma 4.5 *If $s(x) \in \mathcal{P}^{\mathbb{C},\mathbb{Q}}$ say $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$, where $P^s(x) \in \mathbb{Q}[x]$, then all calculations can be performed for the b_i over the integers.*

Proof: To do this, make two observations.

The first observation is that if:

$$b_i = \frac{a_1}{c_1} b_{i-1} + \ldots + \frac{a_m}{c_m} b_{i-m},$$

with a_i, $c_i \in \mathbb{Z}$, then:

$$d^i b_i = \frac{a_1}{c_1} d^i b_{i-1} + ... + \frac{a_m}{c_m} d^i b_{i-m} = \frac{a_1 d}{c_1} d^{i-1} b_{i-1} + ... + \frac{a_m d^m}{c_m} d^{i-m} b_{i-m}.$$

So choose d such that $\frac{a_1 d}{c_1}$, ..., $\frac{a_m d^m}{c_m} \in \mathbb{Z}$. This will give the relation:

$$\bar{b}_i = \bar{a}_1 \bar{b}_{i-1} + ... + \bar{a}_m \bar{b}_{i-m},$$

with $\bar{b}_i = b_i d^i$, and $\bar{a}_i = \frac{a_i d^i}{c_i} \in \mathbb{Z}$.

Notice that the initial values are changed to $\bar{b}_0 = b_0 d^0$, ..., $\bar{b}_m = b_0 d^m$.

The second observations is that if $\bar{b}_0 = \frac{e_0}{f_0}$, ..., $\bar{b}_m = \frac{e_m}{f_m}$, e_i, $f_i \in \mathbb{Z}$ are the initial conditions for the linear recurrence relation then by letting $\bar{d} = \text{lcm}(f_0, ..., f_m)$, the linear recurrence relation:

$$\bar{d}\bar{b}_i = \bar{d}\bar{a}_1 \bar{b}_{i-1} + ... + \bar{d}\bar{a}_m \bar{b}_{i-m},$$

is a calculation made completely over the integers.

■

Example 24 *Consider the following example in Maple.*

```
>  with(MS):
```

Consider the exponential generating function s$(x) = \sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}$, *where b_i satisfy the linear recurrence relation $b_i = \frac{b_{i-1}}{2} + \frac{b_{i-2}}{4}$, with initial conditions of $b_0 = 0$, $b_1 = \frac{1}{3}$. Notice that the computation $bp_i = 2^i\, b_i$ using the linear recurrence relation $2^i\, b_i = 2^{(i-1)}\, b_{i-1} + 2^{(i-2)}\, b_{i-2}$, or equivalently $bp_i = bp_{i-1} + bp_{i-2}$ gives the same result. Remember that now the initial values are $bp_0 = 0$ and $bp_1 = \frac{2}{3}$. Now notice that if instead $bpp_i = 3\, bp_i$ is computed then the computation is wholly within the integers, as are the initial values. So from this it follows that $b_i = \frac{bpp_i}{2^i\, 3}$. Check this by computing the first few terms of both $\frac{bpp_i}{2^i\, 3}$ and b_i.*

```
>  Bpp := `egf/makeproc`(bpp(i) = bpp(i-1) + bpp(i-2), bpp, i,
>  [bpp(0)= 0, bpp(1) = 2]):
>  seq(1/3*(1/2)^i*Bpp(i),i=0..10);
```

$$0, \frac{1}{3}, \frac{1}{6}, \frac{1}{6}, \frac{1}{8}, \frac{5}{48}, \frac{1}{12}, \frac{13}{192}, \frac{7}{128}, \frac{17}{384}, \frac{55}{1536}$$

```
>  B := `egf/makeproc`(b(i) = b(i-1)/2+b(i-2)/4, b, i, [b(0) = 0, b(1) = 1/3]):
>  seq(B(i),i=0..10);
```

$$0, \frac{1}{3}, \frac{1}{6}, \frac{1}{6}, \frac{1}{8}, \frac{5}{48}, \frac{1}{12}, \frac{13}{192}, \frac{7}{128}, \frac{17}{384}, \frac{55}{1536}$$

4.7 Techniques for smaller recurrences.

This section is interested in methods to speed up the calculation of the coefficients of poly-exponential functions. One way, that was suggested by Wilf [30], is to do a calculation of a simpler linear recurrence relation, and then use a non-linear (yet simple) means to get the desired sequence.

This is stated formally as:

Theorem 4.3 *Let $t(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!} \in \mathcal{P}$ have an N-term linear recurrence relation $b_i = \alpha_1 b_{i-1} + \ldots \alpha_N b_{i-N}$. Let $p(x) = \beta_n x^n + \ldots + \beta_0$ be some polynomial in $\mathbb{C}[x]$. Then $p(x)t(x) = \sum_{j=0}^{\infty} d_j \frac{x^j}{j!}$, where $d_i = \beta_n i^{(n)} b_{i-n} + \beta_{n-1} i^{(n-1)} b_{i-n+1} + \ldots \beta_0 b_i$.*

Proof: Then:

$$\begin{aligned}\sum_{j=0}^{\infty} d_j \frac{x^j}{j!} &= p(x)t(x) = p(x)\sum_{i=0}^{\infty} b_i \frac{x^i}{i!} = \sum_{i=0}^{\infty} (\beta_n x^n + \ldots + \beta_0) b_i \frac{x^i}{i!} \\ &= \sum_{i=0}^{\infty} \beta_n b_i \frac{x^{i+n}(i+n)^{(n)}}{(i+n)!} + \ldots + \beta_0 b_i \frac{x^i}{(i)!} = \sum_{i=0}^{\infty} \beta_n b_{i-n} i^{(n)} \frac{x^i}{i!} + \ldots \beta_0 b_i \frac{x^i}{i!}.\end{aligned}$$

■

Example 25 *Consider the following example in Maple.*

```
>  with(MS):
```

Consider the function $\mathrm{s}(x) = (x^2+1)\left(\sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}\right)$, where the b_is are the Fibonacci numbers satisfing $b_i = b_{i-1} + b_{i-2}$ with initial values of $b_0 = 0$ and $b_1 = 1$. This example shows how to determine the linear recurrence relation for $\mathrm{s}(x) = \sum_{i=0}^{\infty} \frac{d_i\, x^i}{i!}$, where the d_i are to be written as functions of the b_i. But this can just be rewritten as $\left(\sum_{i=0}^{\infty} \frac{b_i\, x^{(i+2)}}{i!}\right) + \left(\sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}\right)$, which is just $\left(\sum_{i=0}^{\infty} \frac{b_i\,(i+2)\,(i+1)\,x^{(i+2)}}{(i+2)!}\right) + \left(\sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}\right)$, or in other words $\sum_{i=0}^{\infty} \frac{(b_{i-2}\, i\,(i-1) + b_i)\, x^i}{i!}$. There is a facility in Maple to make procedures with this additional information of the $p(x)$ in Theorem 4.3, (in this case x^2+1).

```
>  t := b(i) = b(i-1) + b(i-2), b, i, [b(0)=0,b(1)=1];
```

$$\mathrm{t} := \mathrm{b}(i) = \mathrm{b}(i-1) + \mathrm{b}(i-2),\ b,\ i,\ [\mathrm{b}(0) = 0,\ \mathrm{b}(1) = 1]$$

```
>  T := `egf/makeproc`(t):
```

```
>  S := `egf/makeproc`(t,i^2+1):
```

Check the first few cases to see if it is correct.

```
>  seq(i*(i-1)*T(i-2)+T(i),i=0..10);
```

$$0, 1, 1, 8, 15, 45, 98, 223, 469, 970, 1945$$

```
>   seq(S(i),i=0..10);
```

$$0, 1, 1, 8, 15, 45, 98, 223, 469, 970, 1945$$

4.8 Conclusions.

The conclusion that are listed in this section are conclusions as to which implemenations are faster, the conclusions are not for which methods are faster. This is because Maple combines a relatively sophisticate code to deal with certain problems, and some very naive methods for others. Hence the implementation of any method in this chapter can be greatly impacted on by the underlying methods used by Maple for certain problems, (for examples, solving linear systems of equations, how it performs resultants, etc).

The different methods that are possible (in combination or otherwise) are:

1. naive method (Chapter 2 Definition 2.6),
2. multiplying recurrence polynomial (Section 4.1),
3. using resultants on recurrence polynomial (Section 4.2),
4. linear algebra, (Section 4.3),
5. linear algebra with symbolic differentiation, (Section 4.4),
6. compression with any of the above methods, (Section 4.5),
7. working over the integers with any of the above methods, (Section 4.6),
8. factoring out a polynomial to reduce the size of the recurrence polynomial with any of the above methods, (Section 4.7).

- Of the first five, methods 4 and 5 are the most efficient. Multisectioning by m for $m > 7000$ are very doable problems.
- The naive method (method 1) is slow, and works poorly for $m > 14$.
- The recurrence polynomial method (method 2) works well for m that is a product of a large number of small primes. In general though, it does not work for large prime values; for primes $m > 43$, it is not really a feasible method.

- The resultant method (method 3), although not as bad as method 1 or 2 is noticeably slower than method 4 or 5. (For the situation of multisectioning the Fibonacci numbers by 1000, method 4 is faster than method 3 by a factor of 20.)
- The compression techniques (method 6) will improve the efficiency of methods 1, 3, 4, or 5, but do little for method 2, (as this method already takes into account the factorization of m). Here it is easy to do problems on the order of 10^5 (when used in combination with method 4).
- Functions rarely meet the criteria for methods 7 and 8 to be used, so they are not of interest.

Chapter 5

Calculations of recurrences for $\mathcal{R}$.

The previous chapter studied methods to determine the lacunary recurrence relations for multisectioned functions in $\mathcal{P}$. This chapter examines techniques for functions in $\mathcal{R}$.

Section 5.1 of this chapter deals with how to multisection the bottom of a rational poly-exponential function, (i.e. perform the necessary multiplication of poly-exponential functions) by looking at the recurrence polynomial and resultants. Section 5.2 looks at two different related methods to perform the multiplication for the bottom linear recurrence relation using fast Fourier transforms and linear algebra. These methods are also extended to determine the top recurrence. How to determine the top linear recurrence relation by using the knowledge about the bottom and about the numbers themselves is examined in Section 5.3. Section 5.4 investigates how symmetries in a poly-exponential function can simplify the calculation of the bottom lacunary recurrence relation. Sections 5.5 and 5.6 investigates two different methods to simplify the problem, by making sure that the work is always done over the integers, or by factoring out polynomials. The last section, Section 5.7 makes some conclusions about which methods are best for which problems.

5.1 Multisectioning recurrence polynomials by resultants.

Given $s(x)$, $t(x) \in \mathcal{P}$, with recurrence polynomials $P^s(x)$, P^t(x), it is difficult to calculate $P^{st}(x)$, the recurrence polynomial of $s(x)t(x)$. This section will demonstrate a method using resultants to perform this calculation.

Combining the results in Lemma 4.1 with the resultant (Definition 4.1) gives:

Lemma 5.1 *Let $s(x)$ and $t(x) \in \mathcal{P}$, where $s(x) = \sum_{i=1}^{n} p_i(x)e^{\lambda_i x}$ and $t(x) = \sum_{j=1}^{m} q_j(x)e^{\mu_j x}$. Then*

$P^{st}(x) | \prod_{i=1,j=1}^{i=n,j=m} (x - \lambda_i - \mu_j)^{\deg(p_i(x))+\deg(q_i(x))} = \mathrm{Res}_y(P^s(x-y), P^t(y))$.

Recall in Section 4.1 that the order in which the calculations were done made a difference in the efficiency of the computation. Here too, the same order is desirable for calculating the linear recurrence relation of $\prod_{i=0}^{m-1} t(x\omega_m^i)$.

Example 26 *Consider the following example in Maple. For more information about the Maple code, see Appendix A. For the Maple code see Appendix E. The Maple code and help files (including information about syntax) are available on the web at [1].*

```
> with(MS):
```

Consider the Genocchi numbers, as defined by Lehmer [19] having an exponential generating function of $\frac{2x}{e^x+1}$. *The calculation of* $\prod_{i=0}^{m-1} (e^{(x\,\omega_m{}^i)} + 1)$, *where* ω_m *is* $e^{(\frac{2\pi I}{m})}$ *is of interest to compute the recurrence of the denominator. Set* $t(x) = e^x + 1$ *and* $s(x) = 2x$. *Assume that this function is to be multisectioned by 4. Then to do this with recurrence polynomials, first find the recurrence polynomial of* $\mathrm{t}(x) = e^x + 1$. *Notice* $deg^d(t(x)) = 0$ *hence* $deg^d(\prod_{i=0}^{m-1} \mathrm{t}(x\,\omega_m{}^i)) = 0$. *This means that the resulting recurrence polynomial may have no multiple roots.*

```
> t := exp(x)+1;
```

$$t := e^x + 1$$

```
> poly := convert_poly(convert_egf(t,f,x));
```

$$poly := x^2 - x$$

Scale this to get the recurrence polynomial of $\mathrm{t}(-x)$ *and then use the resultant to get the result of multiplying the two poly-exponential functions together.*

```
> poly2 := subs(x=-x,poly);
```

$$poly2 := x^2 + x$$

```
> poly3 := resultant(subs(x=x-y,poly),subs(x=y,poly2),y);
```

$$poly3 := (x^2 - x)(x^2 + x)$$

There are no multiple roots, so factor out multiple root, and factor out the leading coefficient.

```
> gcd(poly3, diff(poly3,x), 'poly4'): poly4 := expand(poly4/lcoeff(poly4,x));
```

$$poly4 := x^3 - x$$

Scale this again, to get the recurrence polynomial for $\mathrm{t}(I\,x)\,\mathrm{t}(-I\,x)$*, and then use the resultant to get the result of multiplying the two poly-exponential functions together.*

```
>   poly5 := subs(x=I*x,poly4);
```

$$\mathit{poly5} := -I\,x^3 - I\,x$$

```
>   poly6 := resultant(subs(x=x-y,poly4),subs(x=y,poly5),y);
```

$$\mathit{poly6} := I\,(x^3 - x)\,(-x^4 - 4\,x^2 - 4 - x^6)$$

There will be no multiple roots, so factor out spurious multiple roots, and factor out the leading coefficient..

```
>   gcd(poly6, diff(poly6,x), 'poly7'):  poly7 := expand(poly7/lcoeff(poly7,x));
```

$$\mathit{poly7} := 3\,x^5 + x^9 - 4\,x$$

Now determine the linear recurrence relation.

```
>   convert_rec(poly7,f,x);
```

$$\mathrm{f}(x) = -3\,\mathrm{f}(x-4) + 4\,\mathrm{f}(x-8)$$

Alternatively, the automated function in Maple could have been used.

```
>   `bottom/ms/result`(t,f,x,4);
```

$$\mathrm{f}(x) = -3\,\mathrm{f}(x-4) + 4\,\mathrm{f}(x-8),\ f,\ x,$$
$$[\mathrm{f}(0) = 16,\ \mathrm{f}(1) = 0,\ \mathrm{f}(2) = 0,\ \mathrm{f}(3) = 0,\ \mathrm{f}(4) = -8,\ \mathrm{f}(5) = 0,\ \mathrm{f}(6) = 0,\ \mathrm{f}(7) = 0,\ \mathrm{f}(8) = 72]$$

Which gives the same result.

This example demonstrates how the order in which the resultants are taken is important. Also shown is how the use of the metric $deg^d(t(x))$ can be used to simplify the computation.

5.2 Fast Fourier transforms and linear algebra.

The methods of linear algebra from Section 4.3 needed to know the first $2N + k$ values, where N is the length of the recurrence polynomial, and k is a bound on the multiplicity of the roots. In a practical situation, the calculation of $\prod_{i=1}^{m} t(\omega_m^i x)$ is of interest where $f(x) = \frac{s(x)}{t(x)}$, $s(x)$, $t(x) \in \mathcal{P}$. If $t(x)$ is easy to approximate as a polynomials, then $t(x\omega_m^i)$ is also easy to approximate as a polynomial, via scaling.

Multiplying polynomials can be done quickly via the "*fast Fourier transform*". Maple uses a "*divide and conquer*" method instead of fast Fourier transform, which is still asymptotically better that the naive polynomial multiplication. All of these algorithms can use fast Fourier transform as the basis of polynomial multiplication, but it was deemed beyond the scope of this thesis to implement this method within Maple. See [12] for a proper definition of the divide and conquer and of fast Fourier transform.

Recall in Section 4.1 that the order in which the calculations were done made a difference in the efficiency of the computation. Here too, the same order is desirable for calculating the linear recurrence relation for $\prod_{i=0}^{m-1} t(x\omega_m^i)$. To determine the top linear recurrence relation, the order is not useful, and the polynomials can only be multiplied together in a naive fashion.

The calculation of multisectioning by m, where $m = d_1 d_2 ... d_k$ with $d_i \in \mathbb{Z}$ where $d_i \geq 2$, where an upper bound for $deg^P(\prod_{i=0}^{m-1}(t(x\omega_m^i))$ (from Lemma 2.5), say N and an upper bound for $deg^d(\prod_{i=0}^{m-1} t(x\omega_m^i))$ (from Lemma 2.4), say k, can use two different approaches to determine the new linear recurrence relation.

5.2.1 Fast Fourier transform method 1.

Calculate a polynomial approximation of $t(x)$ to degree $2N + k$, call this $p(x)$. Then iteratively perform:

$$\prod_{i_k=0}^{d_k-1} \cdots \prod_{i_2=0}^{d_2-1} \prod_{i_1=0}^{d_1-1} p(x\omega_{d_1}^{i_1}\omega_{d_1 d_2}^{i_2} \cdots \omega_{d_1 d_2 .. d_k}^{i_k}),$$

by the fast Fourier transform, doing the inner multiplication first, and using scaling for the next level out, etc. Each time a multiplication is done, truncate the polynomial to degree $2N + k$ as any component of the polynomial past that point is not of interest. After this, use linear algebra on the coefficients, to determine what the linear recurrence relation would be. Scaling by a factor of $(2N + k)!$ avoids using rationals in these calculations (assuming $t(x) \in \mathcal{P}^{\mathbb{C},\mathbb{Z}}$).

The problem with this is that the first few multiplications are expensive, as these are dense polynomials of typically large degree.

As a result of implementing this, a bug in Maple was found, which made the original method to scaling very inefficient. This bug had to do with inefficient powering of roots of unity. See Appendix D Section D.1 for more information about this.

Example 27 *Consider the following example in Maple.*

```
> with(MS):
```

When looking at the "Euler numbers" *[2], generated by* $\frac{2}{e^x+e^{(-x)}}$, *the calculation of* $\prod_{i=0}^{m-1} (e^{(x\,\omega_m{}^i)} +$

$\mathrm{e}^{(-x\,{\omega_m}^i)})$, *where* ω_m *is* $\mathrm{e}^{(\frac{2\,\pi\,I}{m})}$ *is of interest. Set* $\mathrm{t}(x) = e^x + \mathrm{e}^{(-x)}$ *and* $\mathrm{s}(x) = 2$*. This example will multisection by 4. An upper bound on the size of the linear recurrence relation is 16 from Lemma 2.5. Also* $deg^d(t(x)) = 0$*, and hence* $deg^d(\prod_{i=0}^{m-1} \mathrm{t}(x\,{\omega_m}^i)) = 0$*. So polynomials of degree 32 needs to be calculated, and then linear algebra is used to determine the result. So first calculate the Taylor series approximation for* $32!\,\mathrm{t}(x)$*, call this* $\mathrm{T}(x)$ *(scaling by* $32!$ *will mean that the calculation will avoid working over the rationals).*

```
>  t := exp(x)+exp(-x);
```

$$\mathrm{t} := e^x + \mathrm{e}^{(-x)}$$

```
>  T := convert(taylor(t,x=0,33),polynom)*32!;
```

$$\begin{aligned}\mathrm{T} := {} & 526261673867387060334436024320000000 \\ & + 263130836933693530167218012160000000\,x^2 \\ & + 21927569744474460847268167680000000\,x^4 \\ & + 730918991482482028242272256000000\,x^6 \\ & + 13052124847901464790040576000000\,x^8 \\ & + 145023609421127386556006400000\,x^{10} \\ & + 1098663707735813534515200000\,x^{12} \\ & + 6036613778768206233600000\,x^{14} + 25152557411534192640000\,x^{16} \\ & + 82197900037693440000\,x^{18} + 216310263257088000\,x^{20} \\ & + 468204033024000\,x^{22} + 848195712000\,x^{24} + 1304916480\,x^{26} \\ & + 1726080\,x^{28} + 1984\,x^{30} + 2\,x^{32}\end{aligned}$$

Now multiply $\mathrm{T}(x)$ *by* $\mathrm{T}(-x)$ *and divide by* $32!$.

```
>  T2 := convert(series(expand(T * subs(x=-x, T)),x,33),polynom)/32!;
```

$$\begin{aligned}\mathrm{T2} := {} & 1052523347734774120668872048640000000 \\ & + 1052523347734774120668872048640000000\,x^2 \\ & + 350841115911591373556290682880000000\,x^4 \\ & + 46778815454878849807505424384000000\,x^6 \\ & + 3341343961062774986250387456000000\,x^8 \\ & + 148504176047234443833350553600000\,x^{10} \\ & + 4500126546885892237374259200000\,x^{12} \\ & + 98903880151338290931302400000\,x^{14} \\ & + 1648398002522304848855040000\,x^{16} \\ & + 21547686307481109135360000\,x^{18} + 226817750605064306688000\,x^{20} \\ & + 1963790048528695296000\,x^{22} + 14230362670497792000\,x^{24} \\ & + 87571462587678720\,x^{26} + 463341071892480\,x^{28} + 2130303778816\,x^{30}\end{aligned}$$

$$+ 8589934592\, x^{32}$$

Now scale this by I, so that the product will give an approximation for $\frac{\mathrm{T}(x)\,\mathrm{T}(-x)\,\mathrm{T}(I\,x)\,\mathrm{T}(-I\,x)}{32!}$.

```
> T3 := convert(series(expand(T2 * subs(x=I*x, T2)),x,33),polynom)/32!;
```

$$\begin{aligned}\mathrm{T3} := {} & 42100933909390964826754881945600000000 \\ & - 140336446364636549422516273152000000000\, x^4 \\ & + 12028838259825989950501394841600000000\, x^8 \\ & - 558015691813850637434408140800000\, x^{12} \\ & + 850573369301509302009200640000\, x^{16} \\ & - 463615482236751442870272000\, x^{20} + 116632052447399903232000\, x^{24} \\ & - 15180906879485214720\, x^{28} + 1125934266580992\, x^{32}\end{aligned}$$

Now collect the coefficients of importance (the non-zero ones).

```
> for i from 0 to 32 by 4 do
> b[i/4] := coeff(T3,x,i)*i!/32!;
> od;
```

$$b_0 := 16$$

$$b_1 := -128$$

$$b_2 := 18432$$

$$b_3 := -1015808$$

$$b_4 := 67633152$$

$$b_5 := -4286578688$$

$$b_6 := 275012124672$$

$$b_7 := -17590038560768$$

$$b_8 := 1125934266580992$$

Now use linear algebra to solve the linear recurrence relation.

```
> `recurrence/solve/linalg`(b, f, x, 4);
```

$$\mathrm{f}(x) = 1024\,\mathrm{f}(x-8) - 48\,\mathrm{f}(x-4)$$

This could also have be done by using the Maple function for this technique

```
>   `bottom/ms/linalg/fft`(t,f,x,4);
```

$$f(x) = 1024\, f(x-8) - 48\, f(x-4),\ f,\ x, [f(0) = 16,\ f(1) = 0,\ f(2) = 0,\ f(3) = 0,\ f(4) = -128,\ f(5) = 0,\ f(6) = 0,\ f(7) = 0,\ f(8) = 18432]$$

Which is the same result.

5.2.2 Fast Fourier transform method 2.

Again, the calculation of interest is

$$\prod_{i_k=0}^{d_k-1} \cdots \prod_{i_2=0}^{d_2-1} \prod_{i_1=0}^{d_1-1} t(x\omega_{d_1}^{i_1}\omega_{d_1 d_2}^{i_2}\ldots\omega_{d_1 d_2 \ldots d_k}^{i_k})$$

with $t(x) \in \mathcal{P}$. Recall that method 1 (Subsection 5.2.1) performed all of these calculations with a large degree polynomial, performing the inner calculations first, and then the next level out, etc. This method differs in that the inner computation is done with a small degree polynomial, the linear recurrence relation for the inner multiplication is then determined with linear algebra, after which the large degree polynomial needed for the next computation is constructed. By scaling out a factor of $(2N+k)!$ each time, (for the various N and k as they apply to each step), can avoid using rationals in these calculations (assuming $t(x) \in \mathcal{P}^{\mathbb{C},\mathbb{Z}}$).

The advantage to this over method 1 is that the polynomials are of small degree near the beginning of the calculation when they are densest. The disadvantage is that linear algebra is repeatedly used.

As a result of implementing this, a bug in Maple was found, which made the original method to scaling very inefficient. This bug had to do with inefficient powering of roots of unity. See Appendix D Section D.1 for more information about this.

As a result of testing this on large examples, some inefficiencies with the factorial function in Maple were discovered. For more information about this, see Appendix D Section D.6.

Example 28 *Consider the following example in Maple.*

```
>   with(MS):
```

Consider the Lucas numbers as defined by Lehmer, [19]. To avoid confusion with the Lucas numbers defined in Graham, Knuth and Patashnik, [16] we will call these Lucas numbers, "Lucas numbers type II". When looking at the Lucas numbers type II generated by $\frac{x\,e^x}{e^{(2\,x)}-1}$*, the calculation of interest is* $\prod_{i=0}^{m-1} (e^{(2\,x\,\omega_m{}^i)} - 1)$*, where* ω_m *is* $e^{(\frac{2\,\pi\,I}{m})}$*. Set* $t(x) = e^{(2\,x)} - 1$ *and* $s(x) = x\,e^x$*. Assume that the* *function is being multisectioned by 4. Notice* $deg^d(t(x)) = 0$, *and hence that* $deg^d(\prod_{i=0}^{m-1} \mathrm{t}(x\,{\omega_m}^i))$ *= 0. Notice that* $deg^P(\mathrm{t}(x))$ *= 2, hence* $deg^P(\mathrm{t}(x)\,\mathrm{t}(-x))$ *is at most 4. So for the first step only a linear recurrence relation to degree 8 is needed. So first calculate the taylor series approximation for* $\mathrm{t}(x)$*, call this* $8!\,\mathrm{T}(x)$ *(scale by* $8!$ *to avoid having to work over the rationals).*

```
> t := exp(2*x)-1;
```

$$\mathrm{t} := e^{(2\,x)} - 1$$

```
> T := convert(taylor(t,x=0,9),polynom)*8!;
```

$$\mathrm{T} := 80640\,x + 80640\,x^2 + 53760\,x^3 + 26880\,x^4 + 10752\,x^5 + 3584\,x^6 + 1024\,x^7 + 256\,x^8$$

Now multiply $\mathrm{T}(x)$ *by* $\mathrm{T}(-x)$ *and divide by 8!.*

```
> T2 := convert(series(expand(T * subs(x=-x,
> T)),x,9),polynom)/8!;
```

$$\mathrm{T2} := -161280\,x^2 - 53760\,x^4 - 7168\,x^6 - 512\,x^8$$

Determine the interesting (non-zero) values.

```
> for i from 0 to 4 do
> b[i] := coeff(T2,x,2*i)*(2*i)!/8!;
> od;
```

$$b_0 := 0$$

$$b_1 := -8$$

$$b_2 := -32$$

$$b_3 := -128$$

$$b_4 := -512$$

Solve this linear recurrence relation.

```
> rec := `recurrence/solve/linalg`(b, f, x, 2);
```

$$rec := \mathrm{f}(x) = 4\,\mathrm{f}(x-2)$$

```
> t2 := rec, f, x, [f(0) = b[0], f(1) = 0,
> f(2) = b[1], f(3) = 0, f(4) = b[2], f(5) = 0,
> f(6) = b[3], f(7) = 0, f(8) = b[4]];
```

$$\mathrm{t2} := \mathrm{f}(x) = 4\,\mathrm{f}(x-2),\ f,\ x, [\mathrm{f}(0) = 0,\ \mathrm{f}(1) = 0,\ \mathrm{f}(2) = -8,\ \mathrm{f}(3) = 0,\ \mathrm{f}(4) = -32,\ \mathrm{f}(5) = 0,\ \mathrm{f}(6) = -128,\ \mathrm{f}(7) = 0,\ \mathrm{f}(8) = -512]$$

Now determine what $deg^P(T2(x))$ and $deg^d(T2(x))$ are, as these will be useful in the calculation.

```
> `egf/metric/P`(t2);
```

$$3$$

```
> `egf/metric/d`(t2);
```

$$0$$

Notice that $deg^P(t2(x)) = 3$, and hence $deg^P(t2(x) * t2(I * x))$ is at most 9. Thus only the first 18 terms of the polynomial approximation needs to be calculated, say $18!\,\mathrm{t2}(x)$ *(scale by* $18!$ *to avoid having to work over the rationals). Call this T2(x).*

```
> Fun := `egf/makeproc`(t2):
> T2 := add (Fun(i)*x^i/i!,i=0..18)*18!;
```

$$\mathrm{T2} := -25609494822912000\,x^2 - 8536498274304000\,x^4 - 1138199769907200\,x^6 - 81299983564800\,x^8 - 3613332602880\,x^{10} - 109494927360\,x^{12} - 2406481920\,x^{14} - 40108032\,x^{16} - 524288\,x^{18}$$

So now multiply $\mathrm{T2}(x)$ *by* $\mathrm{T2}(I\,x)$ *and divide by 18!.*

```
> T3 := convert(series(expand(T2 * subs(x=I*x, T2)),x,19),polynom)/18!;
```

$$\mathrm{T3} := -102437979291648000\,x^4 + 2276399539814400\,x^8 - 14453330411520\,x^{12} + 20374880256\,x^{16}$$

Collect the interesting (non-zero) terms.

```
> for i from 0 to 4 do
> b[i] := coeff(T3,x,4*i)*(4*i)!/18!;
> od;
```

$$b_0 := 0$$

$$b_1 := -384$$

$$b_2 := 14336$$

$$b_3 := -1081344$$

$$b_4 := 66584576$$

Solve this linear recurrence relation.

```
>   rec := `recurrence/solve/linalg`(b, f, x, 4);
```

$$rec := \mathrm{f}(x) = 1024\,\mathrm{f}(x-8) - 48\,\mathrm{f}(x-4)$$

This also could have been done by using the Maple function for this technique

```
>   `bottom/ms/linalg/fft2`(t,f,x,4);
```

$\mathrm{f}(x) = 1024\,\mathrm{f}(x-8) - 48\,\mathrm{f}(x-4),\ f,\ x,$ [
$\mathrm{f}(0)=0,\ \mathrm{f}(1)=0,\ \mathrm{f}(2)=0,\ \mathrm{f}(3)=0,\ \mathrm{f}(4)=-384,\ \mathrm{f}(5)=0,\ \mathrm{f}(6)=0,\ \mathrm{f}(7)=0,\ \mathrm{f}(8)=14336$]

Which is the same result.

It is worth pointing out in this example that fewer terms of the polynomial needed to be worked out. This was because a better bound for $deg^P(\prod_{i=0}^{m-1} t(x\omega_m^i)$ was known as a result of the iteratively calculating $t(x)t(-x)$ and then $t(x)t(-x)t(Ix)t(-Ix)$.

5.3 Using the bottom linear recurrence relation.

The method described in Section 4.3 is easy if $s(x) \in \mathcal{P}$ is known in poly-exponential function form. But there are situations when to explicitly calculate what $s(x)$ is in poly-exponential function form is space consuming and undesirable. For example when trying to determine the top linear recurrence relation of a rational poly-exponential function.

Consider a rational poly-exponential function $\frac{s(x)}{t(x)}$ where $s(x),\ t(x) \in \mathcal{P}$ with $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ and $t(x) = \sum_{j=0}^{\infty} d_j \frac{x^j}{j!}$. Further assume $\frac{s(x)}{t(x)} = \sum_{i=0}^{\infty} c_i \frac{x^i}{i!}$. This gives

$$\sum_{j=s}^{i} \binom{i}{j} d_j c_{i-j} \quad = \quad b_i \tag{5.1}$$

Then if a simple formulae for the d_is and c_is are known, then the b_i can be determined using Equation 5.1. If a bound on the size of the linear recurrence relation for the b_i is known, say N, and a bound for the metric deg^d on the linear recurrence relation for the b_i is known, say k, then only the first $2N+k$ values of b_i need be calculated to determine the linear recurrence relation for the b_i.

Recall from Section 2.4 that typically the linear recurrence relation for multisectioning some q will be the same regardless of the value of q. This can be utilized here by using the process above for the top when multisectioned by m at 0, and then assume that the linear recurrence relation will be the same when multisectioning at other values of q. Hence linear algebra need not be used to determine the linear recurrence relation but instead simply reuse the linear recurrence relation from the first calculation, thus simplifying future calculations immensely.

Example 29 *Consider the following example in Maple.*

```
>   with(MS):
```

This example tries to find the linear recurrence relation for the top of the Euler numbers $\mathrm{f}(x) = \frac{2}{e^x+e^{(-x)}} = \sum_{i=0}^{\infty} \frac{c_i\, x^i}{i!} = \frac{\sum_{i=0}^{\infty} \frac{b_i\, x^i}{i!}}{\sum_{j=0}^{\infty} \frac{d_j\, x^j}{j!}}$ *given the bottom linear recurrence relation, when multisectioning by 4 at 0. As the function is being multisectioned by 4 at 0, then only those* b_i *where* $i = 0 \bmod 4$ *are needed.*

```
>   bot := `bottom/ms/linalg/fft2`(exp(x)+exp(-x),f,x,4);
```

$$bot := \mathrm{f}(x) = 1024\,\mathrm{f}(x-8) - 48\,\mathrm{f}(x-4),\ f,\ x, [\mathrm{f}(0) = 16,\ \mathrm{f}(1) = 0,\ \mathrm{f}(2) = 0,\ \mathrm{f}(3) = 0,\ \mathrm{f}(4) = -128,\ \mathrm{f}(5) = 0,\ \mathrm{f}(6) = 0,\ \mathrm{f}(7) = 0,\ \mathrm{f}(8) = 18432]$$

```
>   Bot := `egf/makeproc`(bot):
```

Now $b_i = \sum_{j=0}^{i} \mathrm{binomial}(i,\, j)\, c_{i-j}\, d_j$ *from Equation 5.1. An upper bound of the number of* b_i *needed as* $4\, 2^3\, 2\, 2 + 2 = 130$ *by Lemma 2.5.*

```
>   F := i -> add(binomial(i,j)*euler(i-j)*`Bot`(j),j=0..i);

Warning, `j` in call to `add` is not local
```

$$F := i \rightarrow \mathrm{add}(\mathrm{binomial}(i,\, j)\, \mathrm{euler}(i-j)\, \mathrm{Bot}(j),\ j = 0..i)$$

```
>   for i from 4 to 130 by 4 do
>   b[i/4] := F(i):
>   od:
>   rec := `recurrence/solve/linalg`(b,f,x,4);
```

$$rec := \mathrm{f}(x) = 625\,\mathrm{f}(x-12) - 611\,\mathrm{f}(x-8) - 13\,\mathrm{f}(x-4)$$

This could have also been discovered by using some of the other built in functions.

```
>   `top/ms/linalg/fft`(2,exp(x)+exp(-x),f,x,4,2);
```

$$f(x) = 625\,f(x-12) - 611\,f(x-8) - 13\,f(x-4),\ f,\ x, [f(0) = 0,\ f(1) = 0,\ f(2) = -16,\ f(3) = 0,\ f(4) = 0,\ f(5) = 0,\ f(6) = 944,\ f(7) = 0,\ f(8) = 0,\ f(9) = 0,\ f(10) = 1904,\ f(11) = 0]$$

```
>   `top/ms/linalg/sym`(2,exp(x)+exp(-x),f,x,4,2);
```

$$f(x) = 625\,f(x-12) - 611\,f(x-8) - 13\,f(x-4),\ f,\ x, [f(0) = 0,\ f(1) = 0,\ f(2) = -16,\ f(3) = 0,\ f(4) = 0,\ f(5) = 0,\ f(6) = 944,\ f(7) = 0,\ f(8) = 0,\ f(9) = 0,\ f(10) = 1904,\ f(11) = 0]$$

This method is automated with the given function below.

```
>   `top/ms/linalg/know`(Bot, euler, f, x, 4, 2, 16, 2);
```

$$f(x) = 625\,f(x-12) - 611\,f(x-8) - 13\,f(x-4),\ f,\ x, [f(0) = 0,\ f(1) = 0,\ f(2) = -16,\ f(3) = 0,\ f(4) = 0,\ f(5) = 0,\ f(6) = 944,\ f(7) = 0,\ f(8) = 0,\ f(9) = 0,\ f(10) = 1904,\ f(11) = 0]$$

Which all give the same result.

Now determine the linear recurrence relation multisectioned by 4 at 2. Taking advantage of the fact of what the linear recurrence relation most likely is, all that really needs to be done is to determine the initial values, and see if the linear recurrence relation is correct. By looking at the recurrence that for the top multisectioned by 4 at 0 that there are only about 12 terms needed. Calculate the first 32 terms for when the function is multisectioned by 4 at 2, and see if this linear recurrence relation holds.

```
>   initial := [seq( op([ f(4*i) = F(4*i), f(4*i+1) = 0, f(4*i+2) = 0, f(4*i+3)
= 0 ]), i=0..8)];
```

$$\mathit{initial} := [f(0) = 16,\ f(1) = 0,\ f(2) = 0,\ f(3) = 0,\ f(4) = -48,\ f(5) = 0,\ f(6) = 0,\ f(7) = 0,\ f(8) = -4208,\ f(9) = 0,\ f(10) = 0,\ f(11) = 0,\ f(12) = 94032,\ f(13) = 0,\ f(14) = 0,\ f(15) = 0,\ f(16) = 1318672,\ f(17) = 0,\ f(18) = 0,\ f(19) = 0,\ f(20) = -77226288,\ f(21) = 0,\ f(22) = 0,\ f(23) = 0,\ f(24) = 257003152,\ f(25) = 0,\ f(26) = 0,\ f(27) = 0,\ f(28) = 44668390992,\ f(29) = 0,\ f(30) = 0,\ f(31) = 0]$$

```
> `egf/clean`(rec, f, x, initial);
```

$$f(x) = 625\,f(x-12) - 611\,f(x-8) - 13\,f(x-4),\ f,\ x, [f(0) = 16,\ f(1) = 0,\ f(2) = 0,\ f(3) = 0,\ f(4) = -48,\ f(5) = 0,\ f(6) = 0,\ f(7) = 0,\ f(8) = -4208,\ f(9) = 0,\ f(10) = 0,\ f(11) = 0]$$

When cleaning up all of the terms, (getting rid of the terms that can be calculated based on the linear recurrence relation) then fewer than the 32 terms are left. Hence, this linear recurrence relation is most probably correct.

This could have done this with the automated function.

```
> `top/ms/know`(rec, Bot, euler, f, x, 4, 0, 130);
```

$$f(x) = 625\,f(x-12) - 611\,f(x-8) - 13\,f(x-4),\ f,\ x, [f(0) = 16,\ f(1) = 0,\ f(2) = 0,\ f(3) = 0,\ f(4) = -48,\ f(5) = 0,\ f(6) = 0,\ f(7) = 0,\ f(8) = -4208,\ f(9) = 0,\ f(10) = 0,\ f(11) = 0]$$

Which gives the same result.

As a result of working on this example, a bug in the help for the Euler function in Maple was found. For more information see Appendix D Section D.2.

5.4 Symmetries.

Recall Lemma 3.1 showed that when multisectioning a rational poly-exponential function $\frac{s(x)}{t(x)}$ by m at q then the bottom poly-exponential function could be written as $\prod_{i=0}^{m-1} t(x\omega_m^i)$ and the top as $(s(x)\prod_{i=1}^{m-1} t(x\omega_m^i))_m^q$. Doing this made the simplifying assumption that there were no common factors among the $t(x\omega_m^i)$, as $0 \le i \le m-1$. For numerous examples of functions, such as the Bernoulli, Euler, Genocchi and Lucas type II numbers, this assumption is not true. (Some rewriting of the Bernoulli and Genocchi functions are needed for this.) This section explores a small subset of the possible situations where this assumption is not valid, and how, by looking at these common factors, the size of the linear recurrence relation can be reduced for the bottom.

These properties have been exploited before in the standard papers on Bernoulli and Euler numbers [9, 19], but, to the best of my knowledge, have not been written in this type of generality before, nor has there been a formal theory behind what is being done.

To this end, define a symmetry.

Definition 5.1 (Symmetry.) *A poly-exponential function, $s(x)$ has a "symmetry of order p" if*

$$s(x\omega_p) = \omega_p^k s(x)$$

for some integer k.

Example 30 *The denominator of the Euler numbers $e^x + e^{-x}$ has a symmetry of order 2.*

Note 5.1 *If $s(x)$ has a symmetry of order p, say $s(x\omega_p) = \omega_p^k s(x)$, then $s(x) = s_p^k(x)$.*

If a symmetry of a function is known, then it can be taken advantage of to find a smaller form for the linear recurrence relation of the denominator of a multisectioned rational poly-exponential function.

Theorem 5.1 *Let $f(x) = \frac{s(x)}{t(x)}$, where $s(x)$, $t(x) \in \mathcal{P}$, and let $t(x)$ have a symmetry of order p, say $t(x\omega_p) = \omega_p^k t(x)$. Further, let $p|m$. Then a recursion formula can be found for the coefficients of x^{mi+q} of the exponential generating function of $f(x)$ that depends only on the coefficients of x^{mj+q}, for $j < i$, and two lacunary recurrence relations, where the lacunary recurrence relation for the denominator has a smaller upper bound on its length than that of Theorem 3.3.*

Proof: Now

$$\begin{aligned}
f_m^q(x) &= \frac{1}{m}\sum_{i=0}^{m-1}\frac{\omega_m^{-iq}s(x\omega_m^i)}{t(x\omega_m^i)} = \frac{1}{m}\sum_{i=0}^{m/p-1}\sum_{j=0}^{p-1}\frac{\omega_m^{-(i+j(m/p))q}s(x\omega_m^{i+j(m/p)})}{t(x\omega_m^{i+j(m/p)})}\\
&= \frac{1}{m}\sum_{i=0}^{m/p-1}\sum_{j=0}^{p-1}\frac{\omega_m^{-iq}\omega_p^{-jq}s(x\omega_m^i\omega_p^j)}{t(x\omega_m^i\omega_p^j)} = \frac{1}{m}\sum_{i=0}^{m/p-1}\sum_{j=0}^{p-1}\frac{\omega_m^{-iq}\omega_p^{-jq}s(x\omega_m^i\omega_p^j)}{\omega_p^{jk}t(x\omega_m^i)}\\
&= \frac{1}{m}\sum_{i=0}^{m/p-1}\sum_{j=0}^{p-1}\frac{\omega_m^{-iq}\omega_p^{-jq-jk}s(x\omega_m^i\omega_p^j)}{t(x\omega_m^i)}\\
&= \frac{1}{m}\sum_{j=0}^{p-1}\sum_{i=0}^{m/p-1}\frac{\omega_m^{-iq}\omega_p^{-jq-jk}s(x\omega_m^i\omega_p^j)\prod_{l=1}^{m/p-1}t(x\omega_m^l)}{\prod_{l=0}^{m/p-1}t(x\omega_m^l)}\\
&= \frac{\frac{1}{m}\sum_{j=0}^{p-1}(\sum_{i=0}^{m/p-1}\omega_m^{-iq}\omega_p^{-jq-jk}s(x\omega_m^i\omega_p^j)\prod_{l=1}^{m/p-1}t(x\omega_m^l))}{\prod_{l=0}^{m/p-1}t(x\omega_m^l)}
\end{aligned}$$

By observing that $t(x) = t_p^k(x)$ a careful analysis shows that $\prod_{l=0}^{m/p-1}t(x\omega_m^l) = (\prod_{l=0}^{m/p-1}t(x\omega_m^l))_m^{km/p}$. Denote this $r_m^{km/p}(x)$. Further, letting $r_m^{km/p}(x) = \sum_{j=0}^{\infty}d_j\frac{x^j}{j!}$, $f(x) = \sum_{i=0}^{\infty}c_i\frac{x^i}{i!}$ and the numerator as $\sum_{i=0}^{\infty}b_i\frac{x^i}{i!}$ gives, from Equation 5.1 that $b_i = 0$ unless $i \equiv q + m/pk \pmod m$. So both the numerator and the denominator are lacunary recurrence relation.

Further, from Lemma 2.5 the denominator $r_m^{mk/p}(x)$ has the property that $deg^P(r_m^{km/p}(x)) \leq deg^P(t(x))^{m/p}$, which is better than the upper bound in Theorem 3.3 of $deg^P(t(x))^m$.

■

Example 31 *Consider the following example in Maple.*

```
> with(MS):
```

Consider the example of the Euler numbers, given by the exponential generating function of $\frac{2}{e^x+e^{(-x)}}$. *The denominator of this has a symmetry of order 2. Below are two methods to compute the recurrence for the denominator, when multisectioned by 8. The first method does not take into account the symmetry, where as the second does. Also demonstrated in this section is the code 'egf/strip', which will strip away the useless zeros.*

```
> botNoSym := `egf/strip`(`bottom/ms/linalg/fft2`(exp(x)+exp(-x),f,x,8,[2,2,2]),
8, 0);
```

$$
\begin{aligned}
\mathit{botNoSym} := {} & \mathrm{f}(x) = -831705558809741310321986973047193 6\,\mathrm{f}(x-80) \\
& + 37233002781512387579098036015464448\,\mathrm{f}(x-72) \\
& + 1166788033962137493268685150748672\,\mathrm{f}(x-64) \\
& - 28599370971194087022785671987 20\,\mathrm{f}(x-56) \\
& - 946119117903717195314311987 2\,\mathrm{f}(x-48) \\
& + 2389168763320088873926656\,\mathrm{f}(x-40) \\
& + 543960885098446848\,\mathrm{f}(x-32) + 3635955734937600\,\mathrm{f}(x-24) \\
& + 158590697472\,\mathrm{f}(x-16) - 283392\,\mathrm{f}(x-8),\ f,\ x, [\mathrm{f}(0) = 256, \\
& \mathrm{f}(8) = -557056,\ \mathrm{f}(16) = 3901315088384,\ \mathrm{f}(24) = -968280866994257920, \\
& \mathrm{f}(32) = 889603035003170066530304, \\
& \mathrm{f}(40) = -391268789233378370377876504576, \\
& \mathrm{f}(48) = 248444193868941930601282703112273920, \\
& \mathrm{f}(56) = -1292153306916561231940897174821658804 87936, \\
& \mathrm{f}(64) = 7459502659938741786901759051414987289821341 2864, \\
& \mathrm{f}(72) = -40726729378210421739875778036712241401761762629386240, \\
& \mathrm{f}(80) = \\
& 229010772884425480073016413254216965235147229465887889 16224]
\end{aligned}
$$

```
> botSym := `egf/strip`(`bottom/ms/linalg/fft2`(exp(x)+exp(-x),f,x,8,[2,2,2],2),
8, 0);
```

$$\mathit{botSym} := \mathrm{f}(x) = -4096\,\mathrm{f}(x-16) - 2176\,\mathrm{f}(x-8),\ f,\ x,\ [\mathrm{f}(0) = 16,\ \mathrm{f}(8) = -17408]$$

```
> BotNoSym := `egf/makeproc`(botNoSym):
> BotSym := `egf/makeproc`(botSym):
```

Next consider the top recurrence, determined by the bottom recurrence and the definition of the Euler numbers, when multisectioning by 8 at 0. Again, the first method does not take into account symmetries, where as the second does.

```
>  topNoSym := `egf/strip`(`top/ms/linalg/know`(BotNoSym, euler, f, x, 8, 0, 30,
   2),8,0);
```

$$\begin{aligned}
&\mathit{topNoSym} := \mathrm{f}(x) = -43920259282210583351533605075940239625113 46\backslash \\
&\quad 48683978210964402620\mathrm{f}(x-112) + 16393772837213378973317\backslash \\
&\quad 938804667469522807655094115550354726553224931 13\mathrm{f}(x-128) - \\
&\quad 679356170320224666233629597717205421703517887823540983 21860 \\
&\quad \mathrm{f}(x-104) + \\
&\quad 2876648532964249458940710162842517424309120851325640780 \\
&\quad \mathrm{f}(x-96) + \\
&\quad 123173556854923811033988111289233893110768422852981 51 \\
&\quad \mathrm{f}(x-88) + 655832062449372229076571004417263593355727800\backslash \\
&\quad 131131068695310939956\mathrm{f}(x-120) + 29932288977535789544 85\backslash \\
&\quad 46471100949079463875894816986365120488249152442430920\backslash \\
&\quad 7\mathrm{f}(x-144) - 215607946609497324829057 02\,\mathrm{f}(x-40) \\
&\quad -114757401756959175156 6\,\mathrm{f}(x-32) + 40165361247172240331\backslash \\
&\quad 342711479814685272767682311113131166993233623934389 41 \\
&\quad \mathrm{f}(x-136) + 785349200709594768478347106782002445343848 91\backslash \\
&\quad 8616770779592494739390443923558731\mathrm{f}(x-152) - 131973\,\mathrm{f}(x-8) \\
&\quad -134667150111\,\mathrm{f}(x-16) \\
&\quad -9517414585447652068034637402058\,\mathrm{f}(x-48) \\
&\quad -92512594457554741735374579001443560538 03\,\mathrm{f}(x-64) \\
&\quad +844986221020858149495600584807102843312837 21\,\mathrm{f}(x-72) \\
&\quad -1133062245492702 7\,\mathrm{f}(x-24) \\
&\quad +23857059979436997763092736685322973457477653881 63\,\mathrm{f}(x-80) \\
&\quad +81302582375740255338429328446321180 6\,\mathrm{f}(x-56) - 534357\backslash \\
&\quad 78925402174043593582652123877017565504064446362041782\backslash \\
&\quad 9564549499574770921434174 1\mathrm{f}(x-192) + 48814666275054200\backslash \\
&\quad 19598847210198941016989441097805143093477702173480496\backslash \\
&\quad 780911470929727\mathrm{f}(x-200) - 16533988700569217371853899 90\backslash \\
&\quad 88841525971187576508195022171625614736231233240285290\backslash \\
&\quad 971\mathrm{f}(x-208) + 232613089138457059038015772154606390984 9\backslash \\
&\quad 3030873003515999259565279964744546250390625\mathrm{f}(x-216) + \\
&\quad 63863245107313263027107180301422790406080328209255828\backslash \\
&\quad 9220903993807817345738772746958\mathrm{f}(x-184) - 320988767861\backslash
\end{aligned}$$

01181638457651707722006068094231238003543924144111985\
708574073397954266f($x - 176$) − 2210912755112381032331783\
37892525477178428723084412946378807494559266311589839\
3462f($x - 168$) − 133606751722998530775061168178227029948\
257269876573785252499938191919945990602718f($x - 160$), f, x, [
f(0) = 256, f(8) = −202496, f(16) = −1063953149696,
f(24) = 64570730111514880, f(32) = 114754084128082385215744,
f(40) = −12617880498158977441699755776,
f(48) = −13558757497291064142754260447399680,
f(56) = 2170619805897092133382221060532917885184,
f(64) = 1558910469676572327193388845250484736038617344,
f(72) = −333883571310415940905401481565768759116901484189440,
f(80) =
−175662840644520683985176861750371976893040536974264594176,
f(88) = 484965667430663900125702569069025106198846760656\
73716295825664, f(96) = 193208469295406084354751329180571\
49348363091224927642362608361529600, f(104) = −6772366215\
58780337958692538975970599262543691319285386324099989\
2141422336, f(112) = −20617726717245267743646468252135139\
63598057011996874738073663893749583225474816, f(120) = 91\
77542784877584642796201877918487801614158162503036098\
66903104438817246864966621440, f(128) = 21143779189020025\
19981142759968877814678251583679412389531928218427761\
85067654949642600704, f(136) = −1213802012475459576770870\
37339337942042861219535152681693440017948231511735765\
368016547875428096, f(144) = −204843433643956974604943432\
60014391790909623367791573352316771152068070097942288\
026991331458997169920, f(152) = 1572454621839193312893023\
47331717651025235741794364808372617318943520862719524\
52387887218455502637116274944, f(160) = 18101793798981768\
97468850458775460758110185354473802874927366725986819\
476313522730216369680662400806527709421824, f(168) = −199\
99867933843650702413266982315969889081786544044255134\
29193831155787180531188119439676780471386768670887694\
728820480, f(176) = −133119618633120701901850512085771108\
32431364491287555186444541707366739308588765758549330\
5736594807846804570292679550799616, f(184) = 250105285632\
03785116103490520672684517978976645058273583097955243\

$$
\begin{aligned}
&88734189966221418884084211412008912595964318134910812\backslash\\
&70300530944, \mathrm{f}(192) = 52588307706405763257356025072347343\backslash\\
&60049528472499715635754362341679820173967167656920852\backslash\\
&960488865900108937646494807733495019412373760, \mathrm{f}(200) = -\backslash\\
&30775725978976969510046824935955938885269811664308452\backslash\\
&69243135131736053830822282944758255863223017288525489\backslash\\
&520566057852221640920045121302 2976, \mathrm{f}(208) = 711146486248\backslash\\
&35393457944380270059054429282827263572950086035928327\backslash\\
&17654936312173082292376933076484311275742660417555667\backslash\\
&52813061990395310739755264]
\end{aligned}
$$

```
>   topSym := `egf/strip`(`top/ms/linalg/know`(BotSym, euler, f, x, 8, 0, 30, 2),8,0);
```

$$
\begin{aligned}
&topSym :=\\
&\quad \mathrm{f}(x) = -6561\,\mathrm{f}(x-32) + 7571428\,\mathrm{f}(x-24) - 45798\,\mathrm{f}(x-16) + 1188\,\mathrm{f}(x-8),\\
&\quad f,\ x,\ [\mathrm{f}(0) = 16,\ \mathrm{f}(8) = 4752,\ \mathrm{f}(16) = 5278992,\ \mathrm{f}(24) = 6144667536]
\end{aligned}
$$

So both the top and the bottom recurrences are smaller when the symmetries of the denominator are taken into account.

5.5 Computing over the integers.

Recall in Section 4.6 that all of the calculations of the coefficients of the exponential generating function of a poly-exponential function can be calculated over the integers if certain criteria are met. Here, a similar result holds, given certain criteria all of the calculations of the coefficients of the exponential generating function of a rational poly-exponential function can be done over the integers.

Consider the equations in Theorem 3.1 again. This gives the following lemma.

Lemma 5.2 *If* $f(x) = \frac{s(x)}{t(x)} = \sum_{i=0}^{\infty} c_i \frac{x^i}{i!}$ *where* $s(x)$, $t(x) \in \mathcal{P}$, *with* $s(x) = \sum_{i=0}^{\infty} b_i \frac{x^i}{i!}$ *and* $t(x) = \sum_{j=0}^{\infty} d_j \frac{x^j}{j!}$ *such that* $d_0 \neq 0$, *where* d_i, $b_i \in \mathbb{Q}$, *and* $P^s(x)$, $P^t(x) \in \mathbb{Q}[x]$ *then all of the calculations of the* c_i *can be done over the integers.*

Proof: A few observations are needed to see this.

Without loss of generality, let $m = 1$ and $q = 0$. Based on the equation of Theorem 3.1 the following equation holds:

$$c_{k-s} = \frac{1}{\binom{k}{s} d_s}(b_k - \sum_{j=s+1}^{k} \binom{i}{j} d_j c_{i-j})$$

Hence, if b_i, $d_i \in \mathbb{Z}$ for all i, $s = 0$, and $d_s = \pm 1$ then $c_i \in \mathbb{Z}$. (This is in fact the case with the Euler numbers.)

Now if $s = 0$, and $d_0 \neq \pm 1$ and $d_0 \in \mathbb{Z}$, then instead calculate $c_i^* = c_i d_0^i$. Notice that:

$$\begin{aligned}
d_0^i c_i &= \frac{d_0^i}{d_0}(b_i - \sum_{j=1}^{i} \binom{i}{j} d_j c_{i-j}) \\
c_i^* &= (d_0^{i-1} b_i - \sum_{j=1}^{i} \binom{i}{j} d_0^{i-1} d_j c_{i-j}) \\
c_i^* &= (d_0^{i-1} b_i - \sum_{j=1}^{i} \binom{i}{j} d_0^{j-1} d_j (d_0^{i-j} c_{i-j})) \\
c_i^* &= (d_0^{i-1} b_i - \sum_{j=1}^{i} \binom{i}{j} d_0^{j} d_j c_{i-j}^*).
\end{aligned}$$

which will remain in the integers.

Further, if b_i and d_i come from functions $s(x)$ and $t(x)$, both of which satisfy all of the conditions of Lemma 4.5, namely that $P^s(x)$, $P^t(x) \in \mathbb{Q}[x]$, where $s(x)$, $t(x) \in \mathcal{P}^{\mathbb{C},\mathbb{Q}}$, then by the c_n^* can be altered so that all the calculations are still done over the integers.

Here take e_b and f_b as the d and c in the proof of Lemma 4.5, as it applies to b_i, and set $\bar{e}_i = b_i e_b^i f_b$. Similarly set $\bar{d}_i = d_i e_d^i f_d$, where e_d and f_d have similar definitions. Further assume that $f_d = 1$.

So now consider calculating $\bar{c}_i = c_i^* \text{lcm}(e_b, e_d)^n \text{lcm}(f_b, f_d)$ For ease of notation, denote $e = \text{lcm}(e_b, e_d)$ and f similarly. For ease of notation, denote $\bar{e}_b = \frac{e}{e_d}$, and define $\bar{e}_d$, $\bar{f}_b$ and $\bar{f}_d$ similarly.

Then:

$$\begin{aligned}
e^i f c_i^* &= e^i f (d_0^i b_i - \sum_{j=1}^{i} \binom{i}{j} d_0^j d_j c_{i-j}^*) \\
\bar{c}_i &= (d_0^i e^i f b_i - \sum_{j=1}^{i} \binom{i}{j} d_0^j e^i f d_j c_{i-j}^*) \\
\bar{c}_i &= (d_0^i (\bar{e}_b)^i \bar{f}_b \bar{b}_i - \sum_{j=1}^{i} \binom{i}{j} d_0^j e^j d_j f e^{i-j} c_{i-j}^*) \\
\bar{c}_i &= (d_0^i (\bar{e}_b)^i \bar{f}_b \bar{b}_i - \sum_{j=1}^{i} \binom{i}{j} d_0^j (\bar{e}_d)^j \bar{d}_j \bar{c}_{i-j}).
\end{aligned}$$

Where finally everything is calculated over the integers.

■

Corollary 10 *The Euler numbers and the Genocchi numbers are integers. Moreover the recursion formula and lacunary recursion formula used to compute the Euler and Genocchi numbers are also over the integers.*

5.6 Techniques for smaller linear recurrence relations.

As before, in Section 4.7, polynomials can be factored from a poly-exponential function, to make the linear recurrence relations easier to solve. Write $t(x) = p(x)\bar{t}(x)$, the denominator of some rational poly-exponential function, for $t(x)$, $\bar{t}(x) \in \mathcal{P}$ and $p(x)$ a polynomials. Then notice, that for calculating the denominator, then a factor of $\prod_{i=0}^{m-1} p(x\omega_m^i)$ can be pulled out.

A similar process for the top linear recurrence relation can be done, but some extra care need be taken.

Example 32 *Consider the following example in Maple.*

```
>  with(MS):
```

This example looks at the Bernoulli numbers. But for this example, modify the equation, so that it can be demonstrated how common factors of polynomials can be taken out. So examine $\frac{x^2+x}{x\,e^x-x+e^x-1} = \frac{\sum_{i=0}^{\infty} \frac{b_i\,x^i}{i!}}{\sum_{j=0}^{\infty} \frac{d_j\,x^j}{j!}}$. *Now multisection this by 4 at 2.*

So the bottom can be

$\prod_{i=0}^{3} (x\,{\omega_4}^i+1)\,(e^{(x\,{\omega_4}^i)}-1) = (\prod_{i=0}^{3} (x\,{\omega_4}^i-1))\,(\prod_{i=0}^{3} (e^{(x\,{\omega_4}^i)}-1))$. *So there is a polynomial that can be factored out. After this simply work out the normal linear recurrence relation for the bottom. This could have done automatically by:*

```
>  `bottom/ms/factor`((x+1)*(exp(x)-1),f,x,4);
```

$$\mathrm{f}(x) = 4\,\mathrm{f}(x-8) - 3\,\mathrm{f}(x-4),\ f,\ x,\ [$$
$$\mathrm{f}(0)=0,\ \mathrm{f}(1)=0,\ \mathrm{f}(2)=0,\ \mathrm{f}(3)=0,\ \mathrm{f}(4)=-24,\ \mathrm{f}(5)=0,\ \mathrm{f}(6)=0,\ \mathrm{f}(7)=0,\ \mathrm{f}(8)=56]$$
$$,\ -x^4+1$$

Where the last value is the polynomial that is pulled out.

The top can be similarly manipulated so as to get the common polynomial to be pulled out.

```
> `top/ms/factor`(x^2+x, (x+1)*(exp(x)-1),f,x,4,2);
```

$$\mathrm{f}(x) = 4\,\mathrm{f}(x-2) - 3\,\mathrm{f}(x-1),\ f,\ x, [\mathrm{f}(0) = 0,\ \mathrm{f}(1) = 0,\ \mathrm{f}(2) = 0,\ \mathrm{f}(3) = 0,\ \mathrm{f}(4) = 0,\ \mathrm{f}(5) = -10,\ \mathrm{f}(6) = 0,\ \mathrm{f}(7) = 0,\ \mathrm{f}(8) = 0,\ \mathrm{f}(9) = 30,\ \mathrm{f}(10) = 0,\ \mathrm{f}(11) = 0,\ \mathrm{f}(12) = 0,\ \mathrm{f}(13) = -130,\ \mathrm{f}(14) = 0,\ \mathrm{f}(15) = 0,\ \mathrm{f}(16) = 0,\ \mathrm{f}(17) = 510,\ \mathrm{f}(18) = 0,\ \mathrm{f}(19) = 0,\ \mathrm{f}(20) = 0,\ \mathrm{f}(21) = -2050,\ \mathrm{f}(22) = 0,\ \mathrm{f}(23) = 0,\ \mathrm{f}(24) = 0,\ \mathrm{f}(25) = 8190,\ \mathrm{f}(26) = 0,\ \mathrm{f}(27) = 0,\ \mathrm{f}(28) = 0,\ \mathrm{f}(29) = -32770,\ \mathrm{f}(30) = 0,\ \mathrm{f}(31) = 0],\ (x-I)\,(x-1)\,(x+I)\,x\,(x+1)$$

5.7 Conclusions.

The conclusion that are listed in this section are conclusions as to which implemenations are faster, the conclusions are not for which methods are faster. This is because Maple combines a relatively sophisticate code to deal with certain problems, and some very naive methods for others. Hence the implementation of any method in this chapter can be greatly impacted on by the underlying methods used by Maple for certain problems, (for examples, solving linear systems of equations, how it performs resultants, etc).

5.7.1 Denominator.

The different methods that are possible for determining the bottom linear recurrence relation of a multisectioned rational poly-exponential function are:

1. naive method, (Chapter 3, Lemma 3.1),
2. the recurrence polynomial with resultants (Section 5.1),
3. linear algebra, with symbolic differentiation (Chapter 4, Section 4.3),
4. linear algebra, fast Fourier transform method 1, (Subsection 5.2.1),
5. linear algebra, fast Fourier transform method 2, (Subsection 5.2.2),
6. looking at symmetries of the denominator, (Section 5.4),
7. computing over the integers, (Section 5.5),
8. factoring polynomials out, in combination with any of the above, (Section 5.6).

- Here, the use of some knowledge (of how large the linear recurrence relation will be) is of great use to method 3 and 4. For example, without this knowledge, trying to determine the bottom linear recurrence relation of the Euler numbers when multisectioned by 8 takes over 60 seconds and 10.65 for methods 3 and 4 respectively, where as with this knowledge this take 4.58 and 3.86 seconds.
- The naive method, method 1, although the easiest to implement, is not very efficient taking 11 seconds to do this problem, whereas method 2 and 5 take 2.72 seconds and 1.42 seconds respectively.
- If the same problem is looked at, but multisectioning by 9 instead of by 8, then of all the methods from 1 to 5, with the exception of method 5, take too long to be practical (even with knowledge).
- Method 5 takes about 126.9 seconds.
- By taking into account a symmetry (method 6) of order p, the existing methods can be expected to be able to multisection by a factor of p more. For example, with the Euler numbers, instead of having a upper bound of 12 for multisectioning, an upper bound of about 24 is achieved. (The Euler numbers have a symmetry of order 2 in the denominator.)
- Methods 7 and 8 are of little interest, as rarely do functions meet the criteria that would be required for these methods to be of use.
- (These times were done on "bb" (2 180 MHZ IP27 Processors, Main memory size, 256 Mbytes), using the Maple interpretation of a CPU second.)

5.7.2 Numerator.

The different methods that are possible for determining the top linear recurrence relation of a multisectioned rational poly-exponential function are:

1. naive method, (Chapter 3, Lemma 3.1),
2. the recurrence polynomial and resultants (Section 5.1),
3. linear algebra with symbolic differentiation, (Chapter 4, Section 4.3),
4. linear algebra, fast Fourier transform, (Subsection 5.2),
5. factoring polynomials out, in combination with any of the above, (Section 5.6),
6. using information about the bottom linear recurrence relation. (Section 5.3).

- Again the problem of the Euler numbers was looked at - trying to determine the top linear recurrence relation.
- An examination of the times gives that method 6 is by far the best.
- When multisectioning by 8 at 2, the other methods, in order take;
  - with method 1, 201.733 seconds,
  - with method 2, over 1000 seconds,
  - with method 3, over 1000 seconds,
  - with method 3, 55.62 (with knowledge),
  - with method 4, over 1000 seconds, and
  - with method 4, 494.15 (with knowledge).
- This is in comparison to method 6, which took only 30.467 seconds.
- If the denominator had a symmetry of order p, then it becomes possible to multisection by a factor of p more. For example, instead of having an upper bound of multisectioning by 12 for the Euler numbers, the upper bound becomes 24. (The Euler numbers have a symmetry of order 2 in the denominator.)
- (These times were done on "bb" (2 180 MHZ IP27 Processors, Main memory size, 256 Mbytes), using the Maple interpretation of a CPU second.)

Chapter 6

Doing the calculation.

When doing calculations, there are numerous things that can be done at the programming level to speed up the calculations. The first two sections, Sections 6.1 and 6.2 talk about methods where concurrence is exploited. The third section, Section 6.3 discusses the largest problems at the time of submission of this thesis that these techniques have been used for. The last section, Section 6.4 discusses some methods of validating the correctness of the results.

The methods in this thesis so far have allowed the calculation of terms of rational poly-exponential functions to be run on m different machines by multisectioning by m. After the problem is divided up by multisectioning, to m different computers, no communication is needed between these computers. The method of multisectioning is limited by the size m, as multisectioning by large m quickly becomes impractical. After multisectioning by m, the computation can only be done on at most m different machines.

This does not mean though that only m different processors can be used. By allowing communication between processors, the problem can be broken up further. The basis of this idea is that to calculate the k-th number, the previous $k-1$ numbers are needed, but not all of them need to be known when the computation is started. When calculating the k-th number, have n other processors working out the $k-1$, $k-2$, ..., $k-n$ numbers. So long as this information is available by the end of the computation there is no problem. Many of the techniques for concurrency used here are described in Snow, [27].

There are two different techniques described here. The first as described in Section 6.1 is in the case with n processors, where all the processors are the same speed (i.e. a dedicated multi-processor machine). This type of problem does not need to worry about load balancing.

The second case, as described in Section 6.2 is that with multiple CPU's, not all of which are the same speed (i.e. a cluster of PCs with different clock speeds). To properly take advantage of the CPUs to their maximum efficiency, more complicated code need be written that will attempt to balance the load. Failing to do this will lead to a computation on n CPU's that is only n times faster than the slowest processor.

6.1 Load balanced code.

6.1.1 Overview.

Assume there are n processors, all of which are the same speed, and the calculations are of well-distributed difficulty (as is the case with rational poly-exponential function), then give every n-th problem to each CPU. At the end of each calculation, the results are communicated to the other processors.

For this problem, the master/slave paradigm is used, as it reduces the number of communication channels that are required. The "*process*" package in Maple was used, which utilized the Unix commands of fork, pipe, wait, block, etc. As a result in implementing this, and preparing the worksheets, numerous bugs in the "*process*" package in Maple were found. For more information see Appendix D Sections D.3, D.4, and D.5.

6.1.2 Details of algorithm.

Assume the program is run with n slaves. Using the master/slave paradigm, have the master tell the slave which calculation to start with, and how large an increment to use. So slave 1 is told to calculate b_1, b_{1+n}, b_{1+2n}, ..., up to some maximum, slave 2 will calculate b_2, b_{2+n}, ..., etc. The slave, when it has done a calculation will tell the master. The master then passes this information on to all of the other $n-1$ slaves.

When the slave needs information, it simply waits for the master to provide this information. This is one of the reasons why in this model it is very important that the slaves are the same speed. If one slaves is slower than the other slaves, then all of these slaves will constantly be waiting for this one slave to complete its calculation before they can continue.

This is summarized below in Figure 6.1.

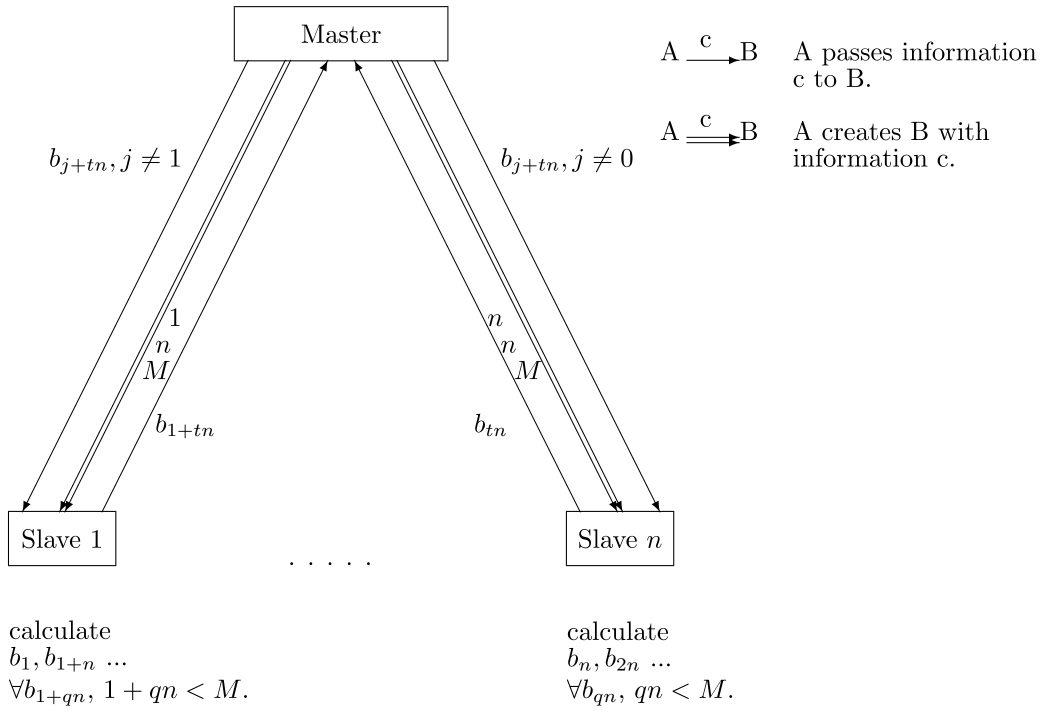

Figure 6.1: Load balanced master/slave diagram.

For more information, see Appendix A, Subsection A.7.2.

Example 33 *Consider the problem of calculating the Genocchi numbers, defined by the exponential generating function $\frac{2x}{e^x+1}$. For more information about the Maple code, see Appendix A. For the Maple code see Appendix E. The Maple code and help files (including information about syntax) are available on the web at [1]. For this, consider the calculation given that the recursion formula is multisectioned by 2 at 0. Further assume that there are two slaves (i.e. a 2 CPU machine).*

```
    |\^/|     Maple V Release 5 (Simon Fraser University)
._|\|   |/|_. Copyright (c) 1981-1997 by Waterloo Maple Inc. All rights
 \  MAPLE  /  reserved. Maple and Maple V are registered trademarks of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(MS): with(process): readlib(`calcul/balanced/worker`):
>
> bot := `bottom/ms/linalg/fft2`(exp(x)+1,f,x,2);
```

```
bytes used=1007116, alloc=851812, time=0.24
          bot := f(x) = f(x - 2), f, x, [f(0) = 4, f(1) = 0, f(2) = 2]

> Bot := `egf/makeproc`(bot):
> top := `top/ms/linalg/fft`(2*x, exp(x)+1, f, x, 2, 0);
top := f(x) = -f(x - 4) + 2 f(x - 2), f, x,

    [f(0) = 0, f(1) = 0, f(2) = -4, f(3) = 0]

> Top := `egf/makeproc`(top):
>
# Increase the information presented, so as to demonstrate how
# the slaves and the master interact with each other.
>
> infolevel[MS] := 4;
                               infolevel[MS] := 4

>
> B := `calcul/balanced`(2, 10, Top, Bot, 2, 0): seq(B[2*i], i=0..5);
calcul/balanced:   "Starting up slave"   0
calcul/balanced/worker:   "Slave"   0   "working on problem"   0
calcul/balanced/worker:   "Slave"   0   "getting needed info from Master"
calcul/balanced/worker:   "Slave"   0   "finishing calculation"
calcul/balanced:   "Starting up slave"   2
calcul/balanced/worker:   "Slave"   0   "Reporting to Master"
calcul/balanced/worker:   "Slave"   2   "working on problem"   2
calcul/balanced/worker:   "Slave"   2   "getting needed info from Master"
calcul/balanced:   "Getting information from slave"   0
calcul/balanced/worker:   "Slave"   0   "working on problem"   4
calcul/balanced/worker:   "Slave"   0   "getting needed info from Master"
calcul/balanced:   "Sending info to slave"   2
calcul/balanced:   "Getting information from slave"   2
calcul/balanced/worker:   "Slave"   2   "finishing calculation"
calcul/balanced/worker:   "Slave"   2   "Reporting to Master"
calcul/balanced/worker:   "Slave"   2   "working on problem"   6
calcul/balanced/worker:   "Slave"   2   "getting needed info from Master"
calcul/balanced:   "Sending info to slave"   0
```

```
calcul/balanced:   "Getting information from slave"   0
calcul/balanced/worker:   "Slave"   0   "finishing calculation"
calcul/balanced/worker:   "Slave"   0   "Reporting to Master"
calcul/balanced/worker:   "Slave"   0   "working on problem"   8
calcul/balanced:   "Sending info to slave"   2
calcul/balanced/worker:   "Slave"   0   "getting needed info from Master"
calcul/balanced/worker:   "Slave"   2   "finishing calculation"
calcul/balanced/worker:   "Slave"   2   "Reporting to Master"
calcul/balanced/worker:   "Slave"   2   "working on problem"   10
calcul/balanced/worker:   "Slave"   2   "getting needed info from Master"
calcul/balanced:   "Getting information from slave"   2
calcul/balanced:   "Sending info to slave"   0
calcul/balanced:   "Getting information from slave"   0
calcul/balanced/worker:   "Slave"   0   "finishing calculation"
calcul/balanced/worker:   "Slave"   0   "Reporting to Master"
calcul/balanced:   "Sending info to slave"   2
calcul/balanced/worker:   "Slave"   2   "finishing calculation"
calcul/balanced/worker:   "Slave"   2   "Reporting to Master"
calcul/balanced:   "Getting information from slave"   2
calcul/balanced:   "Sending info to slave"   0
calcul/balanced:   "Stopping slave"   0
bytes used=1964100, alloc=1441528, time=0.01
calcul/balanced:   "Stopping slave"   2
bytes used=1966624, alloc=1441528, time=0.02
                             0, -1, 1, -3, 17, -155

> quit
bytes used=1969268, alloc=1441528, time=0.51
```

6.2 Load balancing code.

6.2.1 Overview.

If the system does not have balanced CPU power, then the code must balance the load.

Again this method uses the master/slave paradigm, although refinements to this have been made which will be discussed later. Say at some time in the calculation there are k processes running to calculate b_n, b_{n+1}, ..., b_{n+k}. If on the computation $n + s$, $(1 \leq s \leq k)$, the processor can do no more calculations until the information of the value of b_n is provided to it. Instead of waiting (as would have been done in Section 6.1), this process will ask for more work. It will then start calculating b_{n+k+1}, and will get back to the calculations of b_{n+s} when the necessary information is available.

For technical reasons it was decided to have an intermediate process, the overseer, between the master and the slave. This overseer's job is to provide communication between the master and the slave, as well as deciding when a slave can no longer continue working (as the information needed is not available yet), and start a new calculation.

6.2.2 Details of algorithm.

There is one overseer per machine, and one master.

The master will wait until it receives a "need work" message from an overseer. At this point, the master will send the overseer an index of something to be computed.

The overseer will first delegate the work to some slave (if creating the slave, the overseer will also tell the slave everything that the overseer knows).

The slave upon creation/call will start its calculation of the index i given to it. If the slave gets to a point where it needs more information, it will ask the overseer. Upon completion, it will send back the calculation to the overseer and await new work.

The overseer, when it gets a request for information from a slave, will send the information, if it is known. If the information is not known then the overseer will send a message to the master asking for more work. The overseer will keep track that this slave is waiting for this information, and when the overseer acquires this information, it will provide this information to the slave. When the overseer receives the result of a calculation, it will send the result of this calculation to the master. The overseer will ask for work if it has no slaves working (slaves get in each other's way).

The overseer will constantly be waiting for information from the master. The master, when it has a new calculation, will send the information to the other overseers.

This is summarized below in Figure 6.2.

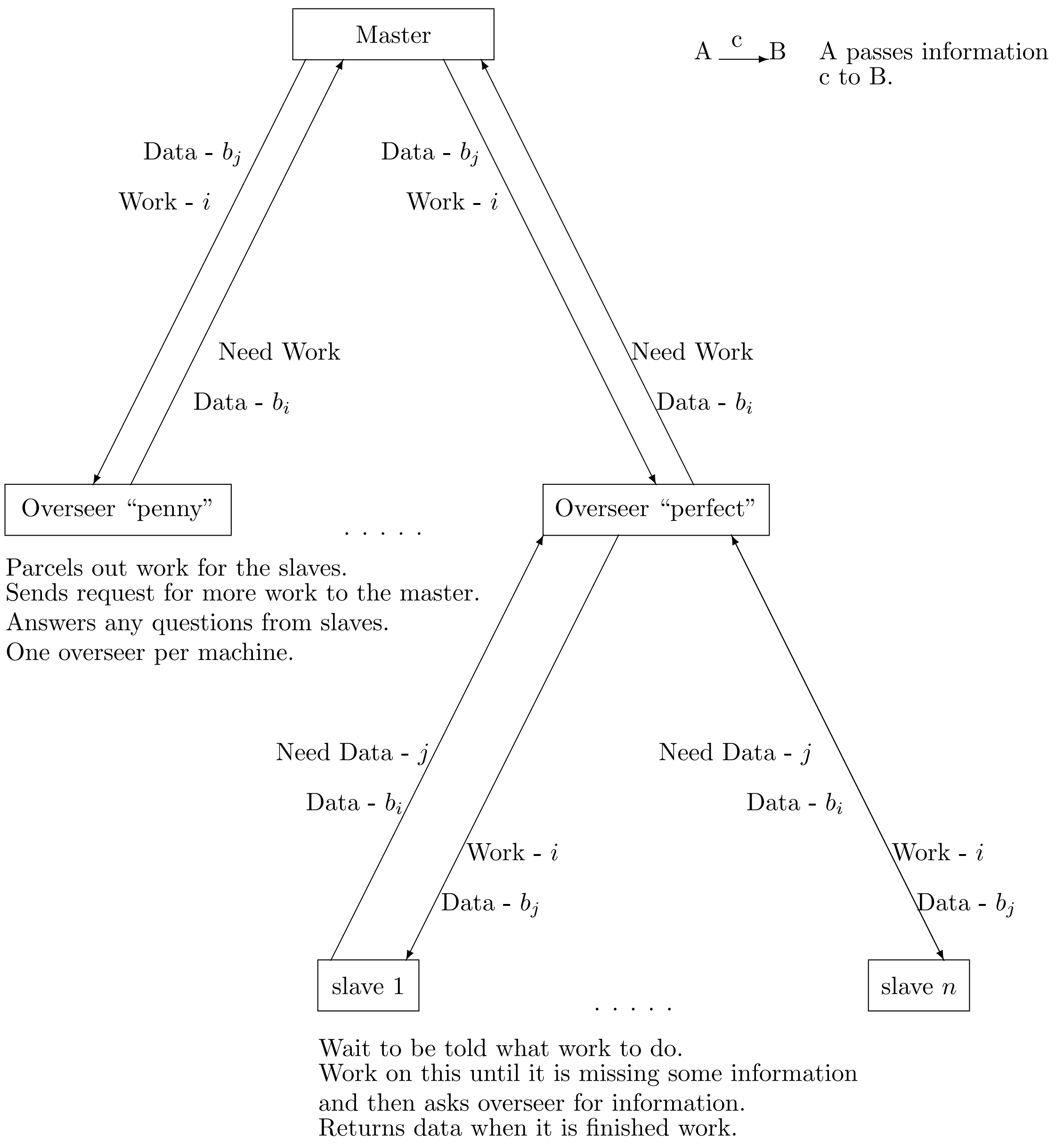
Master
A $\xrightarrow{c}$ B A passes information c to B.
Data - b_j
Work - i
Data - b_j
Work - i
Need Work
Data - b_i
Need Work
Data - b_i
Overseer "penny"
. . . . .
Overseer "perfect"
Parcels out work for the slaves.
Sends request for more work to the master.
Answers any questions from slaves.
One overseer per machine.
Need Data - j
Data - b_i
Work - i
Data - b_j
Need Data - j
Data - b_i
Work - i
Data - b_j
slave 1
. . . . .
slave n
Wait to be told what work to do.
Work on this until it is missing some information
and then asks overseer for information.
Returns data when it is finished work.

Figure 6.2: Load balancing master/overseer/slave diagram.

Example 34 *Consider the following example. The first part is the master, which shows what the* *master is asking the overseer to do. The second and third parts are the two overseers, which demonstrates their side of the conversation.*

1. *The master,*

```
    |\^/|     Maple V Release 5 (Simon Fraser University)
._|\|   |/|_. Copyright (c) 1981-1997 by Waterloo Maple Inc. All rights
 \  MAPLE  /  reserved. Maple and Maple V are registered trademarks of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(MS): with(process):
> Info[0] := 1:
> infolevel[MS] := 2:
> A := `calcul/balancing/master`(bb, [perfect, penny], 10, 2, 2,
>           Euler, 125, Info):
calcul/balancing/master:   "Working on requested for work from perfect"
calcul/balancing/master:   "Tell perfect to work on the value of 2"
calcul/balancing/master:   "Working on requested for work from penny"
calcul/balancing/master:   "Tell penny to work on the value of 4"
calcul/balancing/master:   "Working on requested for work from perfect"
calcul/balancing/master:   "Tell perfect to work on the value of 6"
calcul/balancing/master:   "Working on requested for work from penny"
calcul/balancing/master:   "Tell penny to work on the value of 8"
calcul/balancing/master:   "Got some data for the value of 2 from perfect"
calcul/balancing/master:   "Got some data for the value of 8 from penny"
calcul/balancing/master:   "Working on requested for work from perfect"
calcul/balancing/master:   "Tell perfect to work on the value of 10"
calcul/balancing/master:   "Working on requested for work from penny"
calcul/balancing/master:   "Tell penny to quit"
calcul/balancing/master:   "Got some data for the value of 4 from penny"
calcul/balancing/master:   "Working on requested for work from penny"
calcul/balancing/master:   "Tell penny to quit"
calcul/balancing/master:   "Got some data for the value of 6 from perfect"
calcul/balancing/master:   "Got some data for the value of 10 from perfect"
calcul/balancing/master:   "Telling perfect to quit"
calcul/balancing/master:   "Telling penny to quit"
>
> seq(A[i],i=0..10);
```

```
          1, A[1], -1, A[3], 5, A[5], -61, A[7], 1385, A[9], -50521

> quit
bytes used=420460, alloc=393144, time=0.12
```

2. *Overseer perfect,*

```
    |\^/|     Maple V Release 5 (Simon Fraser University)
._|\|   |/|_. Copyright (c) 1981-1997 by Waterloo Maple Inc. All rights
 \  MAPLE  /  reserved. Maple and Maple V are registered trademarks of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(MS): with(process): readlib(`process/block`):
> readlib(`calcul/writepipe`):
>
> Info[0] := 1:
> Top := `egf/makeproc`(`top/ms/linalg/fft`(2,exp(x)+exp(-x),f,x,2,0)):
> Bot := `egf/makeproc`(`bottom/ms/linalg/fft2`(exp(x)+exp(-x),f,x,2)):
bytes used=1292572, alloc=1048384, time=0.35
>
> infolevel[MS] := 4:
>
> `calcul/balancing/overseer`(bb, perfect, Top, Bot, 2, 0, Info, 1, 1);
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/overseer:
"Has 0 slaves 0 running 0 waiting and the message is Work"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Told to do work on 2 from 0"
calcul/balancing/slave:   "Slave 1 is waiting for instructions"
calcul/balancing/slave:   "Slave 1 is working on determining the value for 2"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Telling the overseer about the new value for 2"
calcul/balancing/slave:   "Slave 1 is waiting for instructions"
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Work"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Told to do work on 6 from 0"
calcul/balancing/overseer:   "Waiting for instructions"
```

```
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Given some new data 2 from 1"
calcul/balancing/overseer:   "Slave"   1   "is no longer working, "
"so give it outstanding work"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Slave 1 is working on determining the value for 6"
calcul/balancing/slave:   "Asking for data of "   2
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Need Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Asked for data"   2   "from"   1
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Got some data 2 from 1"
calcul/balancing/slave:   "Asking for data of "   4
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Need Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Asked for data"   4   "from"   1
calcul/balancing/overseer:   "Doesn't know the info"   4   "for"   1
calcul/balancing/overseer:   "Waiting for instructions"
bytes used=2293024, alloc=1703624, time=1.04
calcul/balancing/overseer:
"Has 1 slaves 1 running 1 waiting and the message is Data"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Given some new data 8 from 0"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/overseer:
"Has 1 slaves 1 running 1 waiting and the message is Work"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Told to do work on 10 from 0"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/overseer:
"Has 1 slaves 1 running 1 waiting and the message is Data"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Given some new data 4 from 0"
```

```
calcul/balancing/overseer:   "Telling waiting slave 1 about this data"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Got some data 4 from 1"
calcul/balancing/slave:   "Telling the overseer about the new value for 6"
calcul/balancing/slave:   "Slave 1 is waiting for instructions"
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Given some new data 6 from 1"
calcul/balancing/overseer:   "Slave"   1   "is no longer working, "
"so give it outstanding work"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Slave 1 is working on determining the value for
10"
calcul/balancing/slave:   "Asking for data of "   6
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Need Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Asked for data"   6   "from"   1
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Got some data 6 from 1"
calcul/balancing/slave:   "Asking for data of "   8
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Need Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Asked for data"   8   "from"   1
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Got some data 8 from 1"
calcul/balancing/slave:   "Telling the overseer about the new value for 10"
calcul/balancing/slave:   "Slave 1 is waiting for instructions"
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Given some new data 10 from 1"
calcul/balancing/overseer:   "Slave 1 is no longer working"
calcul/balancing/overseer:   "Ask for more work"
calcul/balancing/overseer:   "Waiting for instructions"
```

```
calcul/balancing/overseer:
"Has 1 slaves 0 running 0 waiting and the message is Quit"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Telling the 1th slaves to quit"
calcul/balancing/slave:   "Slave Quitting"   1
bytes used=2248948, alloc=1703624, time=0.02
calcul/balancing/overseer:   "The 1th slave has quit"
calcul/balancing/overseer:   "Everyones quit, time to go home"
> quit
bytes used=2600132, alloc=1703624, time=1.30
```

3. *Overseer penny,*

```
    |\^/|     Maple V Release 5 (Simon Fraser University)
._|\|   |/|_. Copyright (c) 1981-1997 by Waterloo Maple Inc. All rights
 \  MAPLE  /  reserved. Maple and Maple V are registered trademarks of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(MS): with(process): readlib(`process/block`):
> readlib(`calcul/writepipe`):
>
> Info[0] := 1:
> Top := `egf/makeproc`(`top/ms/linalg/fft`(2,exp(x)+exp(-x),f,x,2,0)):
> Bot := `egf/makeproc`(`bottom/ms/linalg/fft2`(exp(x)+exp(-x),f,x,2)):
bytes used=1292572, alloc=1048384, time=0.33
>
> infolevel[MS] := 4:
>
> `calcul/balancing/overseer`(bb, penny, Top, Bot, 2, 0, Info, 1, 1);
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/overseer:
"Has 0 slaves 0 running 0 waiting and the message is Data"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Given some new data 8 from 0"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/overseer:
"Has 0 slaves 0 running 0 waiting and the message is Work"
calcul/balancing/overseer:   "Got info from slave/master 0"
```

```
calcul/balancing/overseer:   "Told to do work on 4 from 0"
calcul/balancing/slave:   "Slave 1 is waiting for instructions"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Slave 1 is working on determining the value for 4"
calcul/balancing/slave:   "Asking for data of "   2
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Need Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Asked for data"   2   "from"   1
calcul/balancing/overseer:   "Doesn't know the info"   2   "for"   1
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/overseer:
"Has 1 slaves 1 running 1 waiting and the message is Work"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Told to do work on 8 from 0"
calcul/balancing/overseer:   "Already know the info"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/overseer:
"Has 1 slaves 1 running 1 waiting and the message is Data"
calcul/balancing/overseer:   "Got info from slave/master 0"
calcul/balancing/overseer:   "Given some new data 2 from 0"
calcul/balancing/overseer:   "Telling waiting slave 1 about this data"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/slave:   "Got some data 2 from 1"
calcul/balancing/slave:   "Telling the overseer about the new value for 4"
calcul/balancing/slave:   "Slave 1 is waiting for instructions"
bytes used=2292796, alloc=1572576, time=0.84
calcul/balancing/overseer:
"Has 1 slaves 1 running 0 waiting and the message is Data"
calcul/balancing/overseer:   "Got info from slave/master 1"
calcul/balancing/overseer:   "Given some new data 4 from 1"
calcul/balancing/overseer:   "Slave 1 is no longer working"
calcul/balancing/overseer:   "Ask for more work"
calcul/balancing/overseer:   "Waiting for instructions"
calcul/balancing/overseer:
"Has 1 slaves 0 running 0 waiting and the message is Quit"
calcul/balancing/overseer:   "Got info from slave/master 0"
```

```
calcul/balancing/overseer:   "Telling the 1th slaves to quit"
calcul/balancing/slave:   "Slave Quitting"   1
bytes used=2210520, alloc=1507052, time=0.01
calcul/balancing/overseer:   "The 1th slave has quit"
calcul/balancing/overseer:   "Everyones quit, time to go home"
> quit
bytes used=2346676, alloc=1572576, time=0.89
```

6.3 A large calculation.

As of submitting this thesis, the following upper bounds of calculations have been completed, as shown in the Table 6.1. These calculations are available on the web at [1].

	Bernoulli numbers	Euler numbers	Genocchi numbers	Lucas numbers type II
Bottom recurrence	20	24	20	20
Top recurrence	18	16	20	14
Largest number	35 298	8 500	8 700	5 404

Table 6.1: Upper bounds of completed calculations.

The typical bottle neck for a calculation is with the linear algebra. If a proper Toeplitz matrix solver were used, one would predict that the time to perform a calculation would be much improved. For example, to calculate the denominator of the Bernoulli numbers, multisectioned by 20, it requires only 7 minutes 15 seconds to determine the underlying matrix; the rest of the 2.6 days is to find the solution associated with this 90×90 matrix. (The time here represents a CPU second as measured by Maple on "penny", CPU: MIPS R10000 Processor Chip Revision: 2.7.)

Similarly, when multisectioning the numerator of the Bernoulli numbers by 18 it takes 69.6 seconds to determine the underlying 24×24 matrix and the remained of the 116.35 seconds to solve this linear algebra problem. (The time here represents a CPU second as measured by Maple on "pecos", CPU: MIPS R10000 Processor Chip Revision: 2.7.)

Next consider a large calculation of the Bernoulli numbers, say the first 1 800 Bernoulli numbers. It takes 30.56 seconds to perform this calculation, using recurrences that have been multisectioned by 18. (Hence only $\frac{1}{9}$-th of the information is calculated.) In contrast, the normal recurrence (which by the nature of the Bernoulli numbers is multisectioned by 2) takes 527.61 seconds. Thus there is a speed up of a factor of $\frac{527.61}{30.56 \times 9} = 1.92$ by multisectioning by 18. (Here, the extra factor of 9 comes in because one would have to perform 9 different calculations to get all of the information using the multisectioned method.) This demonstrates that these multisectioned recursion formulae, even when used in serial environment upon a single computer, represent a significant speed up over the traditional recursion formula.

If the multi-processor method described in Section 6.1 is used, with 5 slaves, with the recurrences that has been multisectioned by 18, then it takes on average 6.20 seconds for each slave. (The master takes an insignificant amount of processor time; taking less than half a second.) So the total processor time is bounded above by 31.5 seconds. This indicates that about 3% of the processors time, when using a multi-processor method, goes towards the overhead of communication. (In actual fact, this is too high an estimate when doing a large calculation, but relatively little numerical data is available at this time.) So these calculations can advantageously exploit parallel computing techniques. (The time here represents a CPU second as measured by Maple on "manyjars", 8 250 MHZ IP27 Processors CPU: MIPS R10000 Processor Chip Revision: 3.4.)

6.4 Validating results.

When doing large calculations such as these, some methods to test if the calculations are done correctly are needed, both for confidence and as a useful aid to debugging.

6.4.1 Validating the Bernoulli numbers.

To test if the calculation for the Bernoulli numbers is done correctly, the following theorem of von Staudt [17] is used.

Theorem 6.1 (Clausen - von Staudt Theorem) *Let B_{2k} be the $2k$-th Bernoulli number. If $k \geq 1$, then*

$$(-1)^k B_{2k} \equiv \sum \frac{1}{p} \pmod 1$$

the summation being extended over the primes p such that $(p-1)|2k$.

From which it follows that:

Corollary 11 *If $k \geq 1$, then the denominator of $(-1)^k B_{2k}$, where B_{2k} is the $2k$-th Bernoulli number is equal to the denominator of $\sum \frac{1}{p}$ the summation being extended over the primes p such that $(p-1)|2k$.*

Example 35 *Thus, to test if the 10 008-th Bernoulli number, calculated as*

$$\frac{N}{3262901044146573454170}$$

where N is a 27716 digit number, is correct, the denominator need only be checked.

Calculate $(-1)^k \sum \frac{1}{p}$ for $(p-1)|2k$ where $2k = 10008$ yields:

$$\frac{4402843531608629672099}{3262901044146573454170}$$

Noticing that the denominator of these two numbers is the same is a good indication that the calculation was done correctly.

6.4.2 Validating the Euler numbers.

To test if the calculation for the Euler numbers is done correctly, the following theorem of Glaisher [14] is used.

Theorem 6.2 *Let E_{2k} be the 2k-th Euler number. For $k > 0$, and any $r > 0$:*

$$E_{2k} \equiv (-1)^k 2[1^{2k} - 3^{2k} + 5^{2k} - .... + (-1)^{1/2(r-2)}(r-2)^{2k}] \pmod{r}.$$

Combining this with Fermat's little theorem gives that:

Theorem 6.3 *Let p be prime. If $2k \equiv 2j \pmod{p-1}$ and E_{2k}, E_{2j} the 2k-th and 2j-th Euler numbers respectively then*

$$E_{2k} \equiv E_{2j} \pmod{p}.$$

Example 36 *Thus, to test if the 8 000-th Euler number, calculated as N where N is a 26 184 digit number, is correct, look at N modulo a number of small primes.*

$$\begin{aligned} N &\equiv 2 \pmod{3}, \\ N &\equiv 0 \pmod{5}, \\ N &\equiv 6 \pmod{7}, \\ N &\equiv 2 \pmod{11}, \\ N &\equiv 7 \pmod{13}, \\ N &\equiv 0 \pmod{17}. \end{aligned}$$

Notice that

$$\begin{aligned}
8000 &\equiv 2 \pmod 2 \text{ and } E_2 \equiv 2 \pmod 3,\\
8000 &\equiv 4 \pmod 4 \text{ and } E_4 \equiv 0 \pmod 5,\\
8000 &\equiv 2 \pmod 6 \text{ and } E_2 \equiv 6 \pmod 7,\\
8000 &\equiv 10 \pmod{10} \text{ and } E_{10} \equiv 2 \pmod{11},\\
8000 &\equiv 8 \pmod{12} \text{ and } E_8 \equiv 7 \pmod{13},\\
8000 &\equiv 16 \pmod{16} \text{ and } E_{16} \equiv 0 \pmod{17}.
\end{aligned}$$

Thus N has the correct residues to be the 8 000-th Euler number, and it passes the test.

Chapter 7

Conclusion.

This thesis highlights the complex issues that arise when working in an environment, such as Maple, where the code is not all written by the principle author, or to an agreed standard. One problem in such a system is the necessary reliance on a mixture of code, some of which is very sophisticated, some of which is more naive, some of which is written for a very general problem, and some of which has been tailored to a specific problem. Hence the caveat in Sections 4.8 and 5.7 that the conclusions therein were as to which implementation was fastest, and not to which method was fastest. Another problem is in the debugging of code, where the underlying problem being tracked down in the debugging process might not be within the code written, but instead in the system being used. This could be either an incompatibility of the different functions within the system, a misuse of an algorithm being offered by the system, or an actual problem with the algorithm within the system. Hence the inclusion of Appendix D for bugs or weakness found in Maple.

Some of the achievements of this thesis include implementations of algorithms to multisection rational poly-exponential functions. The new recursion formulae, that these algorithms yield, represent an improvement over the traditional methods of computing Bernoulli numbers, Euler numbers, and other rational poly-exponential functions. Traditionally multisectioning has been looked at in the narrow setting to its use in calculating Bernoulli numbers and Euler numbers. Here, the investigation was done in a more general setting; allowing a wider applicability of the multisectioning process.

Appendix A

Outline of code.

This code can be found on my homepage [1]. It can also be found in Appendix E.

The appendix is laid into five sections. The first section will look at code for manipulating poly-exponential functions. Section A.2 will look at code for manipulating exponential generating functions. Section A.3 looks at the code to determine the metrics of different poly-exponential functions. Section A.4 looks at the code to convert poly-exponential functions to exponential generating functions and back, as well as code to convert linear recurrence relation to the recurrence polynomial and back. Then Section A.5 will look at code for manipulating the bottom linear recurrence relation of a rational poly-exponential function. After which Section A.6 will look at code for manipulating the top linear recurrence relation of a rational poly-exponential function. Lastly Section A.7 will deal with code to do the calculation, after the linear recurrence relations are know.

Within each section, a brief description of a piece of code, the command name, file where it can be found, which example in the thesis demonstrates how it is used with a page reference, the expected input and output of the command, and a reference to which theorems or definitions it automates.

A.1 Code for poly-exponential functions.

A.1.1 Naive method.

This will take a poly-exponential function and multisection it using the naive method, using the definition of multisectioning as given in Definition 2.6.

- file: Pe,

- command: ‘pe/ms/naive‘,
- examples: Example 5 pp. 17,
- input: exponential generating function, m,
- output: exponential generating function multisectioned by m,
- reference: Lemma 2.1, Definition 2.6 and Theorem 2.1.

A.1.2 Linear algebra and symbolic differentiation method.

This method will take a poly-exponential function and multisection it by using symbolic differentiation after which point the method will use linear algebra.

- file: Pe,
- command: ‘pe/ms/linalg/sym‘,
- examples: Example 22 pp. 51,
- input: exponential generating function, $(M, opt), m, q$,
- output: exponential generating function of the poly-exponential function multisectioned by m at q,
- reference: Section 4.3.

A.2 Code for exponential generating functions.

A.2.1 Making procedure from an exponential generating function.

This will turn a linear recurrence relation into a procedure, which will calculate any particular value of the linear recurrence relation.

- file: Egf,
- command: ‘egf/makeproc‘,
- examples: Example 21 pp. 49, Example 24 pp. 58, Example 25 pp. 59, Example 28 pp. 68, Example 29 pp. 72, Example 33 pp. 87, and Example 34 pp. 91,
- input: exponential generating function,
- output: Function.

A.2.2 Stripping zeros from exponential generating function.

This will take a multisectioned exponential generating function, and strip out the terms that are known to be zero.

- file: Egf,
- command: ‘egf/strip‘
- examples: Example 31 pp. 76,
- input: exponential generating function, m, q,
- output: exponential generating function.

A.2.3 Naive method to multisection.

This will take an exponential generating function and multisection it using the naive method as given in Definition 2.6.

- file: Egf,
- command: ‘egf/ms/naive‘,
- examples: Example 5 pp. 17,
- input: exponential generating function, m, q
- output: exponential generating function multisectioned by m, at q,
- reference: Lemma 2.1, Definition 2.6 and Theorem 2.1.

A.2.4 Recurrence polynomial method.

This will take an exponential generating function and multisection it by multiplication of its recurrence polynomial.

- file: Egf,
- command: ‘egf/ms/rec‘,
- examples: Example 19 pp. 45,

- input: exponential generating function, m, q,
- output: exponential generating function multisectioned by m, at q,
- reference: Section 4.1.

A.2.5 Recurrence polynomial via resultants method.

This will take an exponential generating function and multisection it by using resultants.

- file: Egf,
- command: ‘egf/ms/result‘,
- examples: Example 20 pp. 47,
- input: exponential generating function, m, q,
- output: exponential generating function multisectioned by m, at q,
- reference: Section 4.2.

A.2.6 Linear algebra method.

This will take the exponential generating function and use linear algebra to multisection the linear recurrence relation.

- file: Egf,
- command: ‘egf/ms/linalg‘,
- examples: Example 21 pp. 49,
- input: exponential generating function, M, m, q,
- output: exponential generating function multisectioned by m, at q,
- reference: Section 4.3.

A.2.7 Compression method.

This will use compression techniques to multisection the linear recurrence relation of an exponential generating function.

- file: Egf,
- command: ‘egf/ms/compress‘,
- examples: Example 23 pp. 55,
- input: exponential generating function, m, q,
- output: exponential generating function multisectioned by m, at q,
- reference: Section 4.5.

A.3 Metrics.

A.3.1 Metric deg^d.

This is the code that will return $deg^d(s(x))$ given input $s(x)$.

- file: Metric,
- command: ‘egf/metric/d‘, ‘pe/metric/d‘,
- examples: Example 8 pp. 19,
- input: exponential generating function or poly-exponential function,
- output: $deg^d(s(x))$,
- reference: Definition 2.7.

A.3.2 Metric deg^P.

This is the code that will return $deg^P(s(x))$ given input $s(x)$.

- file: Metric,
- command: ‘egf/metric/P‘, ‘pe/metric/P‘,

- examples: Example 8 pp. 19,
- input: exponential generating function or poly-exponential function,
- output: $deg^P(s(x))$,
- reference: Definition 2.7.

A.4 Conversions.

A.4.1 Convert to the recurrence polynomial.

This will convert a linear recurrence relation to a recurrence polynomial.

- file: Convert,
- command: ‘convert_poly‘,
- examples: Example 3 pp. 8,
- input: linear recurrence relation,
- output: recurrence polynomial,
- reference: Definition 2.2.

A.4.2 Convert to the linear recurrence relation.

This will convert a recurrence polynomial to a linear recurrence relation.

- file: Convert,
- command: ‘convert_rec‘,
- examples: Example 3 pp. 8,
- input: recurrence polynomial,
- output: linear recurrence relation,
- reference: Definition 2.2.

A.4.3 Convert to the exponential generating function.

This will convert a poly-exponential function into an exponential generating function so that the linear recurrence relation is easily read.

- file: Convert,
- command: ‘convert_egf‘,
- examples: Example 1 pp. 7,
- input: poly-exponential function,
- output: exponential generating function,
- reference: Lemma 2.1 and Theorem 2.1.

A.4.4 Convert to the exponential generating function.

This will convert an exponential generating function where the linear recurrence relation is easily readable into a poly-exponential function.

- file: Convert,
- command: ‘convert_pe‘,
- examples: Example 2 pp. 8,
- input: exponential generating function,
- output: poly-exponential function,
- reference: Theorem 2.1.

A.5 Bottom linear recurrence relation.

A.5.1 Naive method.

This code will naively use the formula in Lemma 3.1 to determine the bottom linear recurrence relation.

- file: Bottom,

- command: ‘bottom/ms/naive‘,
- examples: Example 13 pp. 30,
- input: poly-exponential function $t(x)$, m,
- output: exponential generating function of $\prod_{i=1}^{m} t(x\omega_m^i)$,
- reference: Lemma 3.1.

A.5.2 Fast Fourier transform and linear algebra.

Uses a combination of linear algebra and fast polynomial multiplication to determine the bottom linear recurrence relation.

- file: Bottom,
- command: ‘bottom/ms/linalg/fft‘, ‘bottom/ms/linalg/fft2‘,
- examples: Example 27 pp. 65 and Example 28 pp. 68,
- input: exponential generating function $t(x)$, M, m,
- output: exponential generating function of $\prod_{i=1}^{m} t(x\omega_m^i)$,
- reference: Section 5.2.

A.5.3 Symbolic differentiation and linear algebra.

This method uses a combination of symbolic differentiation and linear algebra.

- file: Bottom,
- command: ‘bottom/ms/linalg/sym‘,
- examples: Example 22 pp. 51,
- input: poly-exponential function $t(x)$, $2M$, m,
- output: exponential generating function of $\prod_{i=1}^{m} t(x\omega_m^i)$,
- reference: Section 4.3.

A.5.4 Using the recurrence polynomial and resultants.

This will use the resultant to determine the linear recurrence relation.

- file: Bottom,
- command: ‘bottom/ms/result‘,
- examples: Example 26 pp. 63,
- input: exponential generating function $t(x)$, m,
- output: exponential generating function of $\prod_{i=1}^{m} t(x\omega_m^i)$,
- reference: Section 5.1.

A.5.5 Factoring out common polynomials.

This factors out common polynomials to simplify the problem. This can be used in combination with any of the other methods.

- file: Bottom,
- command: ‘bottom/ms/factor‘,
- examples: Example 32 pp. 81,
- input: poly-exponential function $t(x)$, m,
- output: exponential generating function of $(\prod_{i=1}^{m} t(x\omega_m^i))$,
- reference: Section 4.7.

A.6 Top linear recurrence relation.

A.6.1 Naive method.

This code will naively use the formula in Lemma 3.1 to determine the top linear recurrence relation.

- file: Top,
- command: ‘top/ms/naive‘,

- examples: Example 13 pp. 30,
- input: poly-exponential functions $t(x)$, $s(x)$, m, q,
- output: exponential generating function of $(s(x)\prod_{i=1}^{m-1} t(x\omega_m^i))_m^q$,
- reference: Lemma 3.1.

A.6.2 Fast Fourier transform and linear algebra method.

This will use a combination of fast polynomial multiplication and linear algebra to solve the problem.

- file: Top,
- command: ‘top/ms/linalg/fft‘,
- examples: Example 27 pp. 65,
- input: exponential generating function $t(x)$, $s(x)$, M, m, q,
- output: exponential generating function of $(s(x)\prod_{i=1}^{m-1} t(x\omega_m^i))_m^q$,
- reference: Section 5.2.

A.6.3 Symbolic differentiation and linear algebra.

This uses a combination of symbolic differentiation and linear algebra.

- file: Top,
- command: ‘top/ms/linalg/sym‘,
- examples: Example 22 pp. 51,
- input: exponential generating function of $s(x), t(x)$, $\prod t(x\omega_m^i)$, m, q,
- output: exponential generating function of $(s(x)\prod_{i=1}^{m-1} t(x\omega_m^i))_m^q$,
- reference: Section 4.3.

A.6.4 Computing top linear recurrence relation with bottom.

This computes the top linear recurrence relation given the bottom linear recurrence relation.

- file: Top,
- command: ‘top/ms/linalg/know‘,
- examples: Example 29 pp. 72,
- input: exponential generating function of $s(x)$, $t(x)$, $\prod t(x\omega_m^i)$, m, q,
- output: exponential generating function of $(s(x)\prod_{i=1}^{m-1} t(x\omega_m^i))_m^q$,
- reference: Section 5.3.

A.6.5 Knowing probably linear recurrence relation.

This computes the initial values given the top linear recurrence relation, the bottom linear recurrence relation and the recursion formula.

- file: Top,
- command: ‘top/ms/know‘,
- examples: Example 29 pp. 72,
- input: exponential generating function of $s(x)$, $t(x)$, $\prod t(x\omega_m^i)$, m, q,
- output: exponential generating function of $(s(x)\prod_{i=1}^{m-1} t(x\omega_m^i))_m^q$,
- reference: Section 5.3.

A.6.6 Computing new recurrence polynomial using resultants.

This computes the new recurrence polynomial by using resultants.

- file: Top,
- command: ‘top/ms/result‘,
- examples: Example 26 pp. 63,
- input: exponential generating function $s(x)$, $t(x)$, m, q,

- output: exponential generating function of $(s(x)\prod_{i=1}^{m-1} t(x\omega_m^i))_m^q$,
- reference: Section 5.1.

A.6.7 Factoring out common polynomials.

This method will factor out common polynomials to simplify the problem. This can be used in combination with any of the other methods.

- file: Top,
- command: ‘top/ms/factor‘,
- examples: Example 32 pp. 81,
- input: poly-exponential function $s(x)$, $t(x)$,
- output: exponential generating function of $(s(x)\prod_{i=1}^{m-1} t(x\omega_m^i))_m^q$,
- reference: Section 4.7.

A.7 Doing the calculation.

A.7.1 Normal method.

This is just the normal method, using only one processor.

- file: Normal,
- command: ‘calcul/normal‘,
- examples: Example 13 pp. 30,
- input: linear recurrence relations, m, q, and how far to calculate.,
- output: the $mi + q$-th values ,
- reference: Theorem 3.1.

A.7.2 Multiprocessor, even load-balance method.

This will assume multiple, evenly balanced processors, which this algorithm will take advantage of with communication.

- file: Multi,
- command: ‘calcul/balanced‘,
- examples: Example 33 pp. 87,
- input: linear recurrence relations, m, q, and how far to calculate,
- output: the $mi + q$-th values,
- reference: Section 6.1.

A.7.3 Multiprocessor, uneven load-balance method.

This will assume multiple, unevenly balanced processors. This algorithm will balance, and utilize these processors with communication to perform calculations.

- file: Multi,
- command: ‘calcul/balancing‘,
- examples: Example 34 pp. 91,
- input: linear recurrence relation, m, q, and how far to calculate,
- output: the $mi + q$-th values,
- reference: Section 6.2.

Appendix B

Notation.

Symbol,	Meaning,	Page,
α, β,	elements of $\mathbb{C}$,	
γ,	Euler gamma function,	1,
λ, μ,	elements of $\mathbb{C}$ as $e^{\lambda x}$ or $(x-\lambda)$,	
τ,	golden ratio,	1,
ω_m,	root of unity,	10,
$\zeta(n)$,	Riemann zeta function.	1,
a_i, b_i, d_i,	variables in a linear recurrence relation,	6,
c_i,	variables in a recursion formula,	28,
$deg^d(f(x))$,		19,
$deg^P(f(x))$,		19,
$f(x)$, $g(x)$, $h(x)$,	functions in $\mathcal{R}$,	26,
$f_m^q(x)$,	multisectioned function,	10,
i, j, k,	indexes for sums, or products,	
$j^{(r)}$,	$j(j-1)(j-2)...(j-r+1)$	
m,	by what a function is multisectioned,	10,
n,	a fixed integer,	
$p_i(x)$, $q_i(x)$	polynomials in x,	
q	to what a function is multisectioned,	10,
r_i	an unrelated set of integers,	
$r(x)$, $s(x)$, $t(x)$	functions in $\mathcal{P}$,	5,

x	variable,	
y	variable of integration or resultant,	
$\mathbb{C}$	Complex numbers,	
$C_m^q(f_m^q(x))$,	Compression,	53.
G,	Catalan's constant,	1,
N	size of the linear recurrence relation in $\mathcal{P}$,	
$P^f(x)$,	Recurrence polynomial	8,
$\mathcal{P}$,	Poly-exponential functions,	5,
$\mathcal{P}_{R_1,R_2}$,		10,
$\mathcal{P}^{R_1,R_2}$,		10,
$\mathbb{Q}$	Rationals,	
R_i,	subrings of $\mathbb{C}$,	
$\mathcal{R}$,	Rational poly-exponential functions,	26,
$\hat{\mathcal{R}}$,		28,
$\mathcal{R}^{R_1,R_2}$,		33,
$\mathcal{R}_{R_1,R_2}$,		33,
$\hat{\mathcal{R}}^{R_1,R_2}$,		33,
$\hat{\mathcal{R}}_{R_1,R_2}$,		33,
$\mathrm{Res}_x(p(x), q(x))$,	Resultant,	47,
$\mathbb{Z}$	Integers,	

Appendix C

Definitions.

Definition,	Symbol,	Page,
Bernoulli numbers,	$\frac{x}{e^x-1}$	27,
Bernoulli polynomials,	$\frac{xe^{tx}}{e^x-1}$	40,
Catalan's constant,	G,	1,
Chebyshev T polynomials,		23,
Compression,	C_m^q (for some q and m),	53,
Compression by m,	C_m^q (for some q),	53,
Compression by m at q,	C_m^q,	53,
Divide and conquer,		65,
Euler gamma function,	γ,	1,
Euler numbers	$\frac{2}{e^x+e^{-x}}$	65,
Fast Fourier transform		65,
Fibonacci numbers,		8,
Genocchi numbers,	$\frac{2x}{e^x+1}$	63,
Golden mean,	τ,	1,
Lacunary recurrence relation,		11,
Lacunary recursion formula,		30,
Linear recurrence relation,		6,
Lucas numbers type I,		17,
Lucas numbers type II,	$\frac{x}{e^x-e^{-x}}$	68,
Multisection,	$f_m^q(x)$ (for some q and m),	10,

Definition,	Symbol,	Page,
Multisection by m,	$f_m^q(x)$ (for some q),	11,
Multisection by m at q,	$f_m^q(x)$,	11,
Padovan numbers,		47,
Poly-exponential function,	$\mathcal{P}$,	5,
Rational poly-exponential function,	$\mathcal{R}$,	26,
Recursion formula,		28,
Recurrence polynomial,	$P^f(x)$,	8,
Resultant,	$\mathrm{Res}_x(p(x), q(x))$,	47,
Riemann zeta function,	$\zeta(n)$,	1,
Symmetry of order p,		74.

Appendix D

Maple bugs and weaknesses.

This appendix includes some email corresponding between myself and Maple Software concerning bugs and weaknesses in their product. Some editing has been done on the letters for brevity as well as grammatical and spelling corrections.

D.1 Bug 7345 - expand/bigpow and roots of unity.

```
From kghare Thu Nov 26 17:14:46 1998
Subject: expand/bigpow
To: mapledev@daisy.uwaterloo.ca

Why is ‘expand/bigpow‘ being called in the second case?  It is noticeable
slower.

Kevin
-------


kernelopts(printbytes=false);
Poly := convert(taylor(exp(x)-1,x=0,73)*72!,polynom):

readlib(profile);
readlib(‘expand/bigpow‘):

profile(‘expand/bigpow‘):
```

```
tt := time():
poly[2] := expand(subs(x=x*exp(4*Pi*I/5),Poly)):
time() - tt;
showprofile(`expand/bigpow`);

tt := time();
poly[3] := expand(subs(x=x*exp(6*Pi*I/5),Poly)):
time() - tt;
showprofile(`expand/bigpow`);
```

```
> Poly := convert(taylor(exp(x)-1,x=0,73)*72!,polynom):
>
> readlib(profile);
                               proc()  ...  end

> readlib(`expand/bigpow`):
>
> profile(`expand/bigpow`):
>
> tt := time():
> poly[2] := expand(subs(x=x*exp(4*Pi*I/5),Poly)):
> time() - tt;
                                     .054

> showprofile(`expand/bigpow`);
function           depth    calls     time    time%         bytes   bytes%
---------------------------------------------------------------------------
expand/bigpow          0        0    0.000     0.00             0     0.00
---------------------------------------------------------------------------
total:                 0        0    0.000     0.00             0     0.00

>
> tt := time();
                                  tt := .122
```

```
> poly[3] := expand(subs(x=x*exp(6*Pi*I/5),Poly)):
> time() - tt;
                                  12.906

> showprofile(`expand/bigpow`);
function           depth    calls     time    time%         bytes   bytes%
---------------------------------------------------------------------------
expand/bigpow          2     1917   12.877   100.00      37245244   100.00
---------------------------------------------------------------------------
total:                 2     1917   12.877   100.00      37245244   100.00



From kghare Mon Nov 30 15:14:08 1998
Subject: Re: expand/bigpow
To: mapledev@daisy.uwaterloo.ca

I found an easier example demonstrating that something is
wrong.  Noticed, I only changed which 5th root of unity
I was looking at.

> exp(2*Pi*I*2/5)^500;
                                          500
                               exp(4/5 I Pi)

> expand(%);
                                   1

> time();
                                  .079

> exp(2*Pi*I*3/5)^500;
                                           500
                              exp(- 4/5 I Pi)

> expand(%);
```

```
bytes used=1000132, alloc=786288, time=0.19
bytes used=2000888, alloc=1179432, time=0.40
bytes used=3001084, alloc=1441528, time=0.68
bytes used=4001256, alloc=1769148, time=1.00
<SNIP>
bytes used=115099316, alloc=18477768, time=87.06
bytes used=116099516, alloc=18543292, time=88.17
bytes used=117099900, alloc=18739864, time=89.30
bytes used=119406568, alloc=19984820, time=90.15
                                       1

This amount of time, (and for that matter, memory requirements)
doesn't seem reasonable for a problem such as this.

Kevin
```

D.2 Bug 7357 - help for Euler.

Help for the Euler function was wrong.

```
From kghare Tue Dec  8 14:34:42 1998
Subject: Help page for Euler
To: mapledev@daisy.uwaterloo.ca

From the help page for the Euler function we have:

>euler - Euler numbers and polynomials
>
>Calling Sequence:
>     euler(n)
>     euler(n, x)
>
>Parameters:
>     n - a non-negative integer
>     x - an expression
>
>Description:
```

```
>- The function euler computes the nth Euler number, or the nth Euler
>  polynomial in x. The nth Euler number E(n) is defined by the exponential
>  generating function:
>
>
>       2/(exp(t)+exp(-t)) = sum(exp(n)/n!*t^n, n = 0..infinity)

This line should read

        2/(exp(t)+exp(-t)) = sum(E(n)/n!*t^n, n = 0..infinity)

and there should be some description of what E(n,x) is, the nth Euler
polynomial.

Kevin
```

D.3 Bug 7497 - the "process" package.

```
From kghare Thu Oct 15 13:20:35 1998
Subject: Process Package in maple
To: mapledev@daisy.uwaterloo.ca

To: Stefan Vorkoetter;
cc: Maple Dev

I am currently trying to use the "process" package in Maple R5.
For some reason, the new forked processes are having problems
reading the library.

I get the error messages:

Error, (in DoWork) `/maple/mapleR5/lib/process/block.m` is an incorrect or ou\
tdated .m file (rFfn)
> quit
bytes used=239656, alloc=262096, time=0.01
Error, (in DoWork) `/maple/mapleR5/lib/process/block.m` is an incorrect or ou\
tdated .m file (ot3d)
> Error, (in Multi2) invalid subscript selector

This appears to be true on both the CECM machines at Simon Fraser University,
and daisy, at SCG.  If you want to see a copy of the code,
it can be found in my daisy account at
~kghare/Multi2

If you don't have access to daisy, and are interested in seeing
the code, just contact me, and I will mail it to you.  (approx 236 lines)

If I have in my program,
unprotect(block);
block := ....
and simply copy the code in, then everything works fine.
Except that it is an ever-growing list of files that I need to
do this to. (binomial, convert/string, type/odd, fprintf, close, readline, ...)

Any suggestions as to what I might be doing wrong would be appreciated.
I am too unfamiliar with the package to decide if it is a bug, or
I am just using it wrong.

Thanks

Kevin

From kghare Tue Nov 10 17:17:53 1998
Subject: Process Package
To: mapledev@daisy.uwaterloo.ca

When using the "process" package in maple, there is something
strange going on with the libraries and/or kernel after a fork
command.  The child process does not seem to be able to access anything
in the library properly, and I get errors such as:

    |\^/|     Maple V Release 5 (Simon Fraser University)
```

```
> read Multi;
> Multi(3,6);
Error, (in DoWork) could not find `process/block` in the library
Error, (in DoWork) could not find `binomial` in the library
> quit
bytes used=227108, alloc=262096, time=Error, exponent too large
maple: unexpected end of input
> quit
bytes used=227208, alloc=262096, time=Error, exponent too large
maple: unexpected end of input
Error, (in Multi) could not find `process/block` in the library

This is making the code very annoying to use, as I have to use
work-arounds to get around this bug.  (I predefine anything that
the child process will need, so that the child process will not need to
access the library.)  This is in the released version of maple, so it
is not simply a problem of rmaple being a bit out of sync.  Further
it occurs both on the CECM machines (in particular "bb"), and on
the SCG machine (daisy), so it is not a problem with any particular
maple installation.

It would be nice if a patch or fix could be found for this, as I am
using this functionality in my research.

read Multi;
Multi(3,100);

Thanks

Kevin Hare
```

D.4 Bug with "process package" and bytes used message.

```
Subject: process[fork] and bytes used message
To: mapledev@daisy.uwaterloo.ca
Date: Wed, 27 Jan 1999 16:19:28 -0800 (PST)
```

```
When the process[fork] command is called, the options about printing bytes,
or not printing bytes is ignored by either the child of the parent.
(Probably the child.)  Also, the printbytes message is not able to figure
out the time, and returns an error message.  This was done with the
following scripts.

kernelopts(printbytes=false);
with(process):

A := proc()
    local pid;
    kernelopts(printbytes=false);

    pid := fork();

    if pid = 0 then # This is the child
        print("The child has run");
        quit;
    else # This is the parent
        print("The parent has run");
    fi;
    RETURN():
end;

A();

------------------------------------------------------------------
    |\^/|     Maple V Release 5 (Simon Fraser University)

> kernelopts(printbytes=false);
                                     true

> with(process):
> A := proc()
>     local pid;
>     kernelopts(printbytes=false);
>     pid := fork();
>     if pid = 0 then # This is the child
>         print("The child has run");
>         quit;
>     else # This is the parent
>         print("The parent has run");
>     fi;
>     RETURN():
> end;
A := proc()
local pid;
    kernelopts(printbytes = false);
    pid := fork();
    if pid = 0 then print("The child has run"); quit
    else print("The parent has run")
    fi;
    RETURN()
end

> A();
                              "The child has run"

                             "The parent has run"

bytes used=209100, alloc=196572, time=Error, (in A) exponent too large
> quit
> bytes used=209612, alloc=196572, time=Error, exponent too large
maple: unexpected end of input

> quit
bytes used=209184, alloc=196572, time=0.05
```

D.5 Bug with “process” package on xMaple.

```
Subject: process[fork] and xmaple interface
```

```
To: mapledev@daisy.uwaterloo.ca

When using the process[fork] command, I get more than one
thread of execution running.  As is standard, I must "quit"
all but one of these threads before returning control to
the command prompt level.  Unfortunately, if I am using
xmaple, any quit command, from either the child, or the parent
will result in the worksheet exiting.  Hence the following
procedure:

with(process);

A := proc()
    local pid;

    pid := fork();

    if pid = 0 then # This is the child
        print("The child has run");
        quit;
    else # This is the parent
        print("The parent has run");
    fi;
    RETURN():
end;

A();

This will run almost properly on the text based version (modulo the other
bug I just reported), but will terminate the worksheet if it is run under
xmaple (occasionally).

Kevin
```

D.6 Bug 7552 - factorial.

```
Subject: Kernel level factorial is slow, inefficient, and forgetful
To: bugkeeper@maplesoft.com

Below is a very rough version of a factorial function.  It is
written using interpreted maple, where as the built-in versions
is kernel level.  Despite the difference in speed of interpreted
code versus kernel level code, the interpreted version is considerably
faster.

    |\^/|     Maple V Release 5 (Simon Fraser University)

> Fac1 := proc(n)
>     local A;
>     if n < 100 then RETURN (n!)
>     else
>          A := ((n^10-45*n^9+870*n^8-9450*n^7+63273*n^6-269325*n^5+
>              723680*n^4-1172700*n^3+1026576*n^2-362880*n)*`procname`(n-10));
>         RETURN(A);
>     fi;
> end:
>
> tt := time(): Fac1(10000): time() - tt;
bytes used=1005196, alloc=982860, time=0.19
<SNIP>
bytes used=18202916, alloc=4259060, time=3.97
                                     4.013

> tt := time(): 10000!: time() - tt;
                                    11.516

Next, if we add some sort of memory to this function (for example,
here I remember every 100 th value), then the speed is greatly increased
for doing multiple calculations, (yet the memory requirements still
remain low).
```

```
> Fac2 := proc(n)
>     local A;
>     if n < 100 then RETURN (n!)
>     elif (n = 0) mod 10 then
>         A := ((n^10-45*n^9+870*n^8-9450*n^7+63273*n^6-269325*n^5+
>              723680*n^4-1172700*n^3+1026576*n^2-362880*n)*`procname`(n-10));
>         if (n=0) mod 100 then
>             `procname`(n) := A;
>         fi:
>         RETURN(A);
>     else
>         RETURN(`procname`(n-1)*n);
>     fi;
> end:
>
> tt := time():
> for i from 1 to 10000 by 19 do
> Fac2(i):
> od:
<SNIP>
bytes used=100348464, alloc=6945544, time=16.19
> time() - tt;
                                   16.263

>
> tt := time():
> for i from 1 to 10000 by 19 do
> i!:
> od:
bytes used=101348908, alloc=6945544, time=22.80
bytes used=102359904, alloc=6945544, time=115.56
bytes used=103367344, alloc=6945544, time=262.03

Killed as I didn't have the patients to wait.  But it is clear that it
is going to take more than 10 times the amount of time to finish.
(I estimated the time that it would take at around 3000 seconds,
but I don't know exactly.)
```

```
Kevin
```

D.7 Bug 5793 - Multi-argument forget does not work.

```
Subject: Forget forgets more than it should.

According to the help page for forget:

Calling Sequence:
     forget(f,...)
     forget(f,a,b,c,...)

Parameters:
     f      - any name assigned to a Maple procedure
     a, b, c, ... - (optional) specific argument sequence for the function f
     ...    - options

<SNIP>

- forget(f,a,b,c,...) causes the value of f(a,b,c,..) to be ``forgotten''. As
  with the one-argument case, the entry for the argument list a,b,c,... is
  removed from the remember table for f and also from the remember table for
  all functions whose names begin with f/.

<SNIP>

Yet this doesn't even work with the example given in the help page.

> f(x) := 456:
> f(y) := 12:
> f(x),f(y);
                                   456, 12

> forget(f,x);
> f(x),f(y);
```

```
                        f(x), f(y)

It is forgetting too much.

Kevin
```

Appendix E

Code

E.1 Conversions.

File name: Convert.

```
## Notation:
## m.s.   = multisection
## r.p.e. = rational poly-exponential function
## p.e.   = poly-exponential function
## e.g.f. = exponential generating function

macro(`egf/clean` = readlib(`egf/clean`));

# convert_pe
#    This will convert an e.g.f. to a p.e.
# Input: e.g.f.
# Output: p.e.
# References: Theorem 2.1.
`convert_pe` := proc(recur, f, var, init)
    local poly, lambda, n, alpha, Pe, i, deg, Ped, eq, soln, Pez, Eq;

    poly := convert_poly(recur, f, var, init);

    lambda := [solve(poly, var)];
    if has(lambda, RootOf) then
        lambda := map(allvalues, lambda):
    fi:

    n := nops(lambda):

    Pe := 0:
    for i from 1 to n do
        deg := degree(coeff(Pe, exp(var*lambda[i])), var):
        if deg = -infinity then
            deg := 0:
        else
            deg := deg + 1:
        fi:

        Pe := a[i] *exp(lambda[i]*var)*var^deg + Pe:
    od:

    Ped := Pe:

    for i from 0 to nops(init) -1 do
        Pez := subs(var=0,Ped):
        Ped := diff(Pe, var):
        eq[i] := subs(init,f(i)=Pez):
    od:

    Eq := {seq(eq[i],i=0..nops(init)-1)};
    Eq := simplify(Eq):
    soln := solve(Eq);

    Pe := subs(soln, Pe):

    RETURN(Pe, var):
end:

# pe/comb
#    Will take a sequence of p.e. components, and combine
#    ones with common lambda.
# Input: seqn of p.e.
# Output: seqn of p.e.
`pe/comb` := proc(seqn)
    local seqn2, lambda, temp, i;
    userinfo(5,'MS',"Combining lambdas together");
    lambda := {};
    seqn2 := {};
    for i in seqn do
        if member(i[2], lambda) then
            temp := select(proc(x,y)
                if evalb(x[2] = y) then RETURN(true) fi; RETURN(false) end,
                seqn2, i[2]);
            seqn2 := seqn2 minus temp;
            temp := op(temp);
            temp[1] := radnormal(temp[1] + i[1]);
            seqn2 := seqn2 union {[temp[1],temp[2]]};
        else
            lambda := lambda union {i[2]};
            seqn2 := seqn2 union {[i[1],i[2]]};
        fi;
    od;

    RETURN(seqn2);
end:

# convert_egf
#     Takes a p.e. and converts it to an e.g.f.
# Input: p.e.
# Output: e.g.f.
# Reference: Theorem 2.1.
`convert_egf` := proc(seqn, f, var)
    local temp, poly, y, seqn2, size, i, init;

    seqn2 := readlib(`pe/convert`)(seqn,var);

    userinfo(3,'MS',"Combining lambdas");
    seqn2 := readlib(`pe/comb`)(seqn2);

    userinfo(3,'MS',"Creating polynomial");
    poly := mul((var-y[2])^(degree(y[1],var)+1),y=seqn2);

    size := degree(poly,var);
    userinfo(3,'MS',"Expanding polynomial of degree", size);
    poly := radnormal(expand(poly * var));
    userinfo(3,'MS',"Creating Recurrence relation");
    poly := convert_rec(poly,f,var);

    userinfo(3,'MS',"Finding taylor series (to deal with lambda 0)");
    temp := add(y[1]*exp(var*y[2]),y=seqn2);

    init := []:
    for i from 0 to size-1 do
        init := [op(init),f(i)=simplify(subs(var=0,temp))];
      if (i mod 10) = 0 then
        userinfo(3,'MS',`Working on coeff`,i);
      fi;
        temp := expand(diff(temp,var));
    od;

    RETURN(`egf/clean`(poly, f, var, map(radnormal,init,expanded)));
end:

# pe/convert
#     Converts a p.e. to a seqence of p.e.'s.
# Input: p.e.
# Output: sequence of p.e.'s.
`pe/convert` := proc(f,var)
    option remember, system;
    local func, t, combo, p, lambda, alpha, tt, counter;

    userinfo(3, 'MS', "Working on poly-exponential function");
    func := convert(f,exp);
    func := expand(func);
    func := convert(func, exp);
    func := combine(func, exp);
    func := convert(func, exp);

    userinfo(3, 'MS', "Combining exp");
    func := combine(func, exp);

    if type(func, `+`) then
        func := [op(func)];
    else
        func := [f];
    fi;

    userinfo(3, 'MS', "Converting the", nops(func),
        "terms to the correct type");

    counter := 0;
    combo := {};
    for tt in func do
      counter := counter + 1;
      if (counter mod 10) = 0 then
          userinfo(3,'MS',"Working on number", counter);
      fi;

        t := combine(tt,exp);
        p := frontend(degree,[t,var]);
        t := t / var^p;
        t := simplify(convert(t,exp));

        if not has(t, var) then
            alpha := t;
            lambda := 0;
        elif type(t,`*`) then
            lambda := select(has, t, var);
            alpha := t/lambda;
            lambda := op(1,lambda);
            lambda := lambda/var;
        else
            alpha := 1;
            lambda := op(1,t);
            lambda := lambda/var;
        fi;
        combo := [op(combo), [alpha * var^p, lambda]];
    od;
    combo := readlib(`pe/comb`)(combo);
    RETURN(combo);
end:

# convert_poly
#     Converts the e.g.f. to its associate recurrence polynomial
# Input: e.g.f.
# Output: Recurrence polynomial
# Reference: Section 2.3, Definition 2.2.
`convert_poly` := proc(recur, f, var, init)
    local size, temp, poly, i, temp1, VAR, k, egf;

    userinfo(3,'MS',"Converting to polynomial");
    temp1 := expand(rhs(recur));

    temp := {};

    if type(temp1,`+`) then
        for i in temp1 do
            if type(i,`*`) then
                temp := {select(has,i,f)} union temp;
            else
                temp := {i} union temp;
            fi;
        od;
    else
        if type(temp1,`*`) then
            temp := {select(has,temp1,f)} union temp;
        else
            temp := {temp1} union op(temp);
        fi;
    fi;
    temp := map2(op,1, temp);
    temp := subs(var=0,temp);
    temp := min(op(temp));
    size := -temp;

    poly := rhs(recur);

    userinfo(3,'MS',"Creating Recurrence polynomial");
    for i from size+1 by -1 to 1 do
       poly := subs( {f(var-i) = VAR^(size-i)},poly);
    od;
    poly := expand(var^(size+1) - subs (VAR=var,poly)*var);

    userinfo(3,'MS',"Determine size of k");
    if type(init,list) then
        egf := `egf/clean`(recur, f, var, init);
        k := nops(egf[4])-size - 1;
        k := max(k,-1);
    else
        k := -1;
    fi;

    poly := expand(poly * var^k);

    RETURN(poly);
end:

# convert_rec
#     Converts the recurrence polynomial to the recurrence of some e.g.f.
# Input: Recurrence polynomial
# Output: Recurrence
# Reference: Section 2.3, Definition 2.2.
`convert_rec` := proc(Poly, f, var)
    local size, VAR, poly, i;

    poly := Poly;
    size := degree(poly,var);
    userinfo(3,'MS',"Expanding polynomial of degree", size);
    poly := expand(poly * var);
    poly := expand(poly/lcoeff(poly));
    userinfo(3,'MS',"Creating recurrence relation");
    for i from size+1 by -1 to 1 do
       poly := subs( {var^i=f(VAR-size+i-1)},poly);
    od;
    poly := subs (VAR=var,poly);
    poly := f(var) = solve(poly,f(var));

    RETURN(poly);
end:

savelib(`convert_pe`, `convert_pe.m`);
savelib(`convert_egf`, `convert_egf.m`);
savelib(`pe/convert`, `pe/convert.m`);
savelib(`convert_poly`, `convert_poly.m`);
savelib(`convert_rec`, `convert_rec.m`);
savelib(`pe/comb`, `pe/comb.m`);
```

E.2 Metrics.

File name: Metric.

```
## Notation:
## m.s.   = multisection
## r.p.e. = rational poly-exponential function
## p.e.   = poly-exponential function
## e.g.f. = exponential generating function

macro(`pe/convert` = readlib(`pe/convert`),
      `pe/comb`    = readlib(`pe/comb`)):

# pe/metric/d
#     Takes a p.e. $s$ and computes $deg^d(s)$
# Input: p.e.
# Output: $deg^d(p.e.)$
# Reference: Definition 2.7.
`pe/metric/d` := proc(pe, var)
    local seqn;
    userinfo(5,'MS',"Determining the maximal degree polynomial of the".
        " poly-exponential function.");
    seqn := [op(expand(pe))];
    seqn := subs(exp=1,seqn);
    seqn := simplify(seqn);
    seqn := map(degree, seqn, var);
    RETURN(max(op(seqn)));
end:

# pe/metric/P
#     Takes a p.e. $s$ and computes $deg^P(s)$
# Input: p.e.
# Output: $deg^P(p.e.)
# Reference: Definition 2.7
`pe/metric/P` := proc(pe, var)
    local seqn, i, P, x;
    userinfo(5,'MS',"Determining the size of the recurrence relationship of ".
                "the poly-exponential function");
    seqn := `pe/convert`(pe, var);
    seqn := `pe/comb`(seqn);
    seqn := [op(seqn)];
    seqn := map(proc(x, var) RETURN(degree(x[1],var)) end, seqn,var);
    P := 1;
    for i in seqn do
        P := P + (i+1);
    od;
    RETURN(P-1);
end:

# egf/metric/d
#     Takes a e.g.f. $s$ and computes $deg^d(s)$
# Input: e.g.f.
# Output: $deg^d(e.g.f.)
# Reference: Definition 2.7
`egf/metric/d` := proc(recur, f, var, init)
    local poly, poly2, i, g;
    userinfo(5,'MS',"Determining the maximal degree polynomial of the ".
        "exponential generating function");
    poly := convert_poly(recur, f, var, init);
    i := 0;
    poly2 := diff(poly,var);
    g := gcd(poly, poly2);
    while g <> 1 do
        i := i + 1;
        poly := g;
        poly2 := diff(poly,var);
        g := gcd(poly, poly2);
    od;
    RETURN(i);
end:

# egf/metric/P
#     Takes a e.g.f. $s$ and computes $deg^P(s)$
# Input: e.g.f.
# Output: $deg^P(e.g.f.)
# Reference: Definition 2.7
`egf/metric/P` := proc(recur, f, var, init)
    local poly;
    userinfo(5,'MS',"Determining the size of the recurrence relationship of ".
                "the exponential generating function");
    poly := convert_poly(recur, f, var, init);
    RETURN(degree(poly,var));
end:

savelib(`egf/metric/d`, `egf/metric/d.m`);
savelib(`egf/metric/P`, `egf/metric/P.m`);
savelib(`pe/metric/d`, `pe/metric/d.m`);
savelib(`pe/metric/P`, `pe/metric/P.m`);
```

E.3 Poly-exponenial function.

File name: Pe.

```
## Notation:
## m.s.   = multisection
## r.p.e. = rational poly-exponential function
## p.e.   = poly-exponential function
## e.g.f. = exponential generating function

macro(`egf/clean` = readlib(`egf/clean`));

# pe/ms/naive
#     M.s. the p.e. by $m$ at $q$ using the naive approach.
# Input: p.e., m, q
# Output: e.g.f.
# Reference: Definiton 2.6.
#           Appendix A.1.1.
`pe/ms/naive` := proc(func, f, var, m, q)
    local pe, egf, k;

    pe := func;

    # Ref: Definition 2.6.
    userinfo(1,'MS',"Multisectioning poly-exponential function");
    pe := 1/m*sum(subs(var=var*(-1)^(2*k/m),pe)*(-1)^(-2*k*q/m),k=1..m);
    userinfo(1,'MS',"Convering multisectioned poly-exponential function to".
        " an exponential generating function.");
    egf := convert_egf(pe, f, var);
    RETURN(`egf/clean`(egf));
end:

# pe/ms/linalg/sym
#     Here we determine the first $2 M m$ initial values (via
#     symbolic differentiation), and then use linear algebra
#     to solve the recurrence relationship.
# Reference: Section 4.3.
#            Appendix A.1.2.
`pe/ms/linalg/sym` := proc(func, f, var, m, q, N)
    local C, MM, Ff, rec, i, initial, FF, Zero, B;

    Zero := `pe/metric/d`(func, var);
    if nargs = 6 then
        MM := N;
    else
        MM := `pe/metric/P`(func,var);
    fi;

    userinfo(1, 'MS', "Taking derivatives to determine taylor-series coeff");
    if q <> 0 then
        Ff := combine(expand(diff(func,[var$q])),exp);
    else
        Ff := combine(func,exp);
    fi;
    C[0] := eval(Ff,var=0);
    for i from 1 to 2 * MM do
        Ff := combine(expand(diff(Ff,[var$m])),exp);
        C[i] := radnormal(eval(Ff,var=0));
    od;

    B := [seq(C[i],i=ceil((Zero-q)/m)..2*MM)];

    userinfo(1, 'MS', "Using linear algebra to determine recurrence of size",
            2*MM);
    rec := `recurrence/solve/linalg`(B, f, var, m);

    FF := proc(i, m, q, C)
        if (i = q) mod m then
            RETURN(C[(i-q)/m]);
        else
            RETURN(0);
        fi;
    end;

    initial := [seq(f(i)=FF(i,m,q, C), i=0..MM * m - 1)];

    RETURN(`egf/clean`(rec, f, var, initial));
end:

# pe/ms
#     M.s. the p.e. by $m$ at $q$.
# Input: p.e., m, q, method[methodarg]
# Output: e.g.f.
`pe/ms` := proc(pe, f, var, m, q, opt)
    local i, method, methodarg, egf;

    userinfo(1, 'MS', "Multisectioning the poly-exponential function.");

    if nargs = 6 then
        if type(opt, indexed) then
            method := `pe/ms/`.(op(0, opt));
            methodarg := op(1, opt);
        else
            method := `pe/ms/`.opt;
        fi;
    else
        method := `pe/ms/linalg/sym`;
    fi;

    if assigned(methodarg) then
        egf := method(pe,f,var,m,q,methodarg);
    else
        egf := method(pe,f,var,m,q);
    fi;

    RETURN(`egf/clean`(egf));
end:

#libname := libname[3], libname[1..2]:

savelib(`pe/ms`, `pe/ms.m`);
savelib(`pe/ms/linalg/sym`, `pe/ms/linalg/sym.m`);
savelib(`pe/ms/naive`, `pe/ms/naive.m`);
```

E.4 Exponential generating function.

File name: Egf.

```
## Notation:
## m.s.   = multisection
## r.p.e. = rational poly-exponential function
## p.e.   = poly-exponential function
## e.g.f. = exponential generating function

macro (clean      = readlib(`egf/clean`),
       ifactors   = readlib(ifactors),
       forget     = readlib(forget),
       compress   = readlib(`egf/compress`),
       y          = `egf/ms/variable/y`,
       nn         = `egf/makeproc/variable/nn`,
       uncompress = readlib(`egf/uncompress`));

# egf/ms/naive
#     M.s. the e.g.f. using the naive method of converting it
#     to a p.e., and then m.s.'ing that using the definition of
#     m.s.
# Input: e.g.f., m , q
# Output: e.g.f.
# Reference: Definition 2.6.
#            Appendix A.2.3.
`egf/ms/naive` := proc(recur, f, var, init, m, q)
    local pe, egf;

    userinfo(1,'MS',"Convering the exponential generating function".
        " to a poly-exponential function".
        " and multisection it");
    pe := convert_pe(recur, f, var, init)[1];
    egf := `pe/ms/naive`(pe, f, var, m, q);

    RETURN(clean(egf));
end:

# egf/ms/result
#     M.s. the e.g.f by looking at the recurrence polynomial, and
#     using resultants.
# Input: e.g.f, m, q
# Output: e.g.f.
# Reference: Section 4.2.
#            Appendix A.2.5.
`egf/ms/result` := proc(recur, f, var, init, m, q)

    local poly, rep, size;

    size := `egf/metric/P`(recur,f,var,init);

    # The maximum number of repeated roots.
    rep  := `egf/metric/d`(recur,f,var,init);

    # Ref Lemma 2.5.
    userinfo(1,'MS', "Creating recurrence polynomial");
    poly := convert_poly(recur,f,var,init);
    size := size * m;

    # Section 4.2.
    userinfo(1,'MS', "Using resultants with the polynomial");
    poly := resultant(subs(var=y,poly), y^m - var^m, y);

    userinfo(1,'MS', "Creating recurrence equation");
    poly := convert_rec(poly,f,var);
    poly := simplify(poly);

    RETURN(clean(poly,f,var,readlib(`egf/init`)(recur,f,var,init,size/m,m,q)));
end:

# egf/ms/rec
#     M.s. the e.g.f. by looking at the recurrence polynomial, and
#     dealing with it in an approriate manner.
# Input: e.g.f., m, q
# Output: e.g.f.
# Reference: Section 4.1.
#            Appendix A.2.4.
`egf/ms/rec` := proc(recur, f, var, init, m, q)

    local poly, size, rep;

    size := `egf/metric/P`(recur,f,var,init);

    # The maximum number of repeated roots.
    rep  := `egf/metric/d`(recur,f,var,init);

    # Ref Lemma 2.5.
    userinfo(1,'MS', "Creating recurrence polynomial");
    poly := convert_poly(recur,f,var,init);
    size := size * m;

    # Section 4.1.
    userinfo(1,'MS', "Multisection recurrence polynomial");
    poly := readlib(`egf/ms/rec/multi`)(poly, var, m, 1, rep);

    userinfo(1,'MS', "Creating recurrence equation");
    poly := convert_rec(poly,f,var);
    poly := simplify(poly);

    RETURN(clean(poly,f,var,readlib(`egf/init`)(recur,f,var,init,size/m,m,q)));
end:

# egf/ms/rec/multi
#     M.s. the recurrence polynomial
# Input: poly, m
# Output: poly
# Reference: Section 4.1.
`egf/ms/rec/multi` := proc(f, x, m, d, rep)
    local p, F, i, F2, G;

    userinfo(3, 'MS', "Using multiplication of recurrence to get ".
                     "the new multisectioned recurrence", d);
    F := 1;
    # Ref: Section 4.1.
    if isprime(m/d) then
       for i from 0 to m/d-1 do
           F := expand(F * subs(x=x*(-1)^(2*i*d/m),f));
       od;
    else
       p  := ifactors(m/d)[2][1][1];

       if nargs = 5 then
           F2 := `procname`(f,x,m,d*p, rep);
       else
           F2 := `procname`(f,x,m,d*p);
       fi;
       for i from 0 to p-1 do
           F := expand(F * subs(x=x*(-1)^(2*i*d/m),F2));
       od;
    fi;

    if nargs = 5 then
        G := F;
        for i from 0 to rep do
            G := gcd(diff(G,x), G);
        od;
        F := quo(F, G, x);
    fi;

    F := expand(F / lcoeff(F, x)):

    RETURN(radnormal(F));
end:

# egf/ms/compress
#     m.s. the e.g.f. by repeated m.s.'ing by prime factor,
#     compressing that result, and m.s.'ing again.  method
#     used to m.s. the e.g.f. will default to linalg, but
#     can be choosed to be something else.
# Input: e.g.f., m, q, (optional) method
# Output: e.g.f.
# Reference: Section 4.5.
#            Appendix A.2.7.
`egf/ms/compress` := proc(recur, f, var, init, m, q, opt, opt2)
    local method, d, p, q1, q2, egf;

    if nargs >= 7 then
        method := `egf/ms/`.opt;
    else
        method := `egf/ms/linalg`;
    fi;


    userinfo(1, 'MS', "Multisection the exponential generating function".
        " using compression techniques and", method);

    egf := recur, f, var, init;

    d := 1;
    q1 := 0;
    q2 := 0;
    p := 1:

    # Ref: Section 4.5.
    while d <> m do
        p := ifactors(m/d)[2][1][1];
        userinfo(2, 'MS', "Calculating multisectioning by", d, "at", q2);
        d := d * p;
        q1 := ((q mod d)-q2)/d*p;
        q2 := q2 + d * q1/p;

        egf := method(egf,p,q1);
        if d = m then break; fi;
        egf := compress(egf, p, q1);
    od;
    if nargs = 8 and opt2 = "Leave Compressed" then
        RETURN(clean(compress(egf,p,q1))):
    fi;

    if m <> p then
        egf := uncompress(egf, m/p, q-m/p*q1);
    fi;

    RETURN(clean(egf));
end:

# egf/ms/linalg
#     M.s. the e.g.f. determining how large the recurrence polynomial
#     is and then calculating even $m$th term and using
#     linear algebra to determine the new recurrence
# Input: e.g.f., m, q
# Output: e.g.f.
# Reference: Section 4.3.
#            Appendix A.2.6.
```

```
`egf/ms/linalg` := proc(recur, f, var, init, m, q)
    local C, MM, Ff, rec, i, initial, FF, Zero;

    MM := `egf/metric/P`(recur, f, var, init);
    Zero := `egf/metric/d`(recur, f, var, init);

    userinfo(1,'MS',"Make the procedure for the egf");
    Ff := `egf/makeproc`(recur, f, var, init);

    for i from Zero to 2 * MM do
        C[i] := Ff(m*i+q);
    od;

    C := convert(C, list):

    userinfo(1,'MS',"Solve new recurrence using linear algebra");
    rec := `recurrence/solve/linalg`(C, f, var, m);

    FF := proc(i, m, q, Ff)
        if (i = q) mod m then
            RETURN(Ff(i));
        else
            RETURN(0);
        fi;
    end;

    initial := [seq(f(i)=FF(i, m, q, Ff), i=0..MM * m - 1)];

    RETURN(clean(rec, f, var, initial));
end:

# egf/ms
#     M.s. the e.g.f. by $m$ at $q$.
# Input: e.g.f., m, q, method[methodarg]
# Output: e.g.f.
`egf/ms` := proc(recur, f, var, init, m, q, opt)
    local i, method, methodarg, egf;

    userinfo(1, 'MS', "Multisectioning the egf");

    if nargs = 7 then
        if type(opt, indexed) then
            method := `egf/ms/`.(op(0, opt));
            methodarg := op(1, opt);
        else
            method := `egf/ms/`.opt;
        fi;
    else
        method := `egf/ms/linalg`;
    fi;

    if assigned(methodarg) then
        egf := method(recur, f, var, init, m, q, methodarg);
    else
        egf := method(recur, f, var, init, m, q);
    fi;

    RETURN(clean(egf));
end:


# egf/clean
#     Will look at the initial conditions and get rid of terms at the
#     end which are not required.
# Input: e.g.f.
# Output: e.g.f.
# Reference: NONE
`egf/clean` := proc(recur, f, var, init)
    local Init, k, Recur, Value;
    option system, remember;
    userinfo(5,'MS',"Getting rid of useless initial values");
    Init := init;
    k := nops(Init);
    do
        Recur := subs(var=k-1, recur);
        Value := subs(init, Recur);
        Value := simplify(lhs(Value)-rhs(Value));
        if evalb(Value=0) then
            Init := Init[1..-2];
            k := k-1;
        else
            break;
        fi;
    od;
    RETURN(recur, f, var, Init, args[5..nargs]);
end:

# egf/makeproc
#     This, given and e.g.f. and a function name, will return a recurrsive
#     function using the recurrence relationship of the e.g.f. and
#     the initial values given.
# Input: e.g.f.
# Output: procedure
# Reference: Appendix A.2.1.
`egf/makeproc` := proc(recur, f, var, init, scale)

    local maxinit, P, Rec, Procname, T, m, n;

    userinfo(1,'MS',"Making the procedure to calculate a recurrence");

    maxinit := map(lhs,init);
    maxinit := map2(op,1,maxinit);
    maxinit := max(op(maxinit));

    Rec := rhs(recur);
    if Rec = NULL then Rec := 0; fi;
    Rec := subs({var=nn, f=Procname}, Rec);
    P := subs({REC=Rec,Init=init,MaxInit=maxinit, F= f},
        (proc(`egf/makeproc/variable/nn`)
            option remember, system;
            if `egf/makeproc/variable/nn` < 0 then
                RETURN(0);
            elif `egf/makeproc/variable/nn` <= MaxInit then
                RETURN(subs(Init,F(`egf/makeproc/variable/nn`)));
            else
                RETURN(REC);
            fi;
        end));

    # This is a hack suggested by Greg Fee to allow me
    # to get the key word "procname" substituted into the
    # procedure, as uneval quotes won't work.
    P := subs(Procname=procna.me,op(P));

    if nargs = 4 then
        RETURN(op(P));
    else
        T := add(coeff(scale,var,m)*expand(i!/(i-m)!)*'P'(i-m),
            m=0..degree(scale,var));
        RETURN(unapply(T,i));
    fi;
end:

# egf/makeproc2
#     This, given and e.g.f. and a function name, will return a recurrsive
#     function using the recurrence relationship of the e.g.f. and
#     the initial values given.
# Input: e.g.f.
# Output: procedure
# Reference: NONE (Yet)
`egf/makeproc2` := proc(recur, f, var, init, After, PROCNAME)

    local maxinit, P, Rec, Procname, T, m;

    userinfo(1,'MS',"Making the procedure to calculate a recurrence");

    maxinit := map(lhs,init);
    maxinit := map2(op,1,maxinit);
    maxinit := max(op(maxinit));

    Rec := rhs(recur);
    if Rec = NULL then Rec := 0; fi;
    Rec := subs({var=nn, f=Procname}, Rec);
    P := subs({REC=Rec,Init=init,MaxInit=maxinit, F= f, after=After,P=PROCNAME},
        (proc(`egf/makeproc/variable/nn`)
            option system, remember;
            if `egf/makeproc/variable/nn` < 0 then
                RETURN(0);
            elif `egf/makeproc/variable/nn` <= MaxInit then
                RETURN(subs(Init,F(`egf/makeproc/variable/nn`)));
            else
                forget(P, `egf/makeproc/variable/nn`-after);
                RETURN(REC);
            fi;
        end));

    # This is a hack suggested by Greg Fee to allow me
    # to get the key word "procname" substituted into the
    # procedure, as uneval quotes won't work.
    P := subs(Procname=procna.me,op(P));

    RETURN(op(P));
end:

# egf/scale
#     Scale an e.g.f. by lambda
# Input: e.g.f., lambda
# Output: e.g.f.
# Reference: NONE
`egf/scale` := proc(recur, f, x, init, lambda)
    local poly, Recur, Init, i;

    userinfo(5,'MS', "Finding P^{f(lambda x))} given P^f and P^g");
    poly := convert_poly(recur,f,x,init);
    poly := subs(x=x/lambda,poly);

    Recur := simplify(expand(convert_rec(poly,f,x)));

    userinfo(5,'MS', "Finding inital values for P^{f(lambda x))} ".
                    "given P^f and P^g");
    Init := [];
    for i in init do
        Init := [op(Init), op(1,i) = expand(op(2,i)*lambda^op([1,1],i))];
    od;
    Init := (expand(radnormal(Init)));

    # Note, do not "clean" these results.
    RETURN(Recur, f, x, Init);
end:


# egf/compress
#     Compress an e.g.f. by $m$ at $q$
# Input: e.g.f., $m$, $q$
# Output: e.g.f.
# Reference: Section 4.5.
`egf/compress` := proc(recur, f, x, init, m, q)

    local Recur, Init, i, F;

    userinfo(3, 'MS', "Working on compressing recurrence");
    Recur := subs([seq(f(x-m*i)=F(x-i),i=0..nops(rhs(recur)))],recur);
    Recur := subs(f = 0 ,Recur);
    Recur := subs(F = f ,Recur);

    Init := map(proc(x, mm, q, init) local i;
        subs([seq(i=(i-q)/mm, i=0..nops(init))],lhs(x)) = rhs(x);
        end, init, m, q, init);
    Init := simplify(Init);
    Init := select(proc(eq) type(op([1,1],eq), integer) end, Init);

    RETURN(clean(Recur, f, x, Init));
end:

# egf/uncompress
#     Uncompress an e.g.f. by $m$ at $q$
# Input: e.g.f., $m$, $q$
# Output: e.g.f.
# Reference: Section 4.5.
`egf/uncompress` := proc(recur, f, var, init, m, q)

    local i, egf, Init, F, j;

    egf := [clean(recur,f,var,init)];

    userinfo(3, 'MS', "Working on uncompressing recurrence");
    egf[1] := subs([seq(var-i=var-m*i,i=1..`egf/metric/P`(op(egf)))],egf[1]);

    Init := [];
    for j from 0 to nops(egf[4])-1 do
        Init := [op(Init),seq(F(i+j*m)=0,i=0..q-1), F(q+j*m)=f(j),
            seq(F(i+j*m)=0,i=q+1..m-1)]
    od;
    Init := subs(egf[4],Init);
    Init := subs(F=f,Init);

    RETURN(clean(egf[1], egf[2], egf[3], Init));
end:

# egf/init
#    Determine the first values up to $N$ of the
#    function for every $m$th value starting at $q$.
# Input: e.g.f., N, m, q
# Output: list
# Reference: NONE
`egf/init` := proc(recur, f, var, init, N, m, q)
    local b, Init, i, s;
    userinfo(4,'MS',"Find initial values for a recurrence");

    if not type(init[1],`=`) then RETURN(init); fi;

    b := `egf/makeproc`(recur, f, var, init);

    if nargs > 5 then
        Init := [seq(seq(f(m*i+s)=Heaviside(s-q+1/2)*
                Heaviside(q-s+1/2)*b(m*i+s),s=0..m-1),i=0..N)];
    else
        Init := [seq(f(i)=b(i),i=0..N)];
    fi;

    RETURN(expand(radnormal(Init)));
end:

# egf/result
#    Determine the resultant of two e.g.f.'s.
# Input: e.g.f. 1, e.g.f. 2
# Output: e.g.f.
# Reference: NONE
`egf/result` := proc(recur1, f1, x1, init1, recur2, f2, x2, init2)
    local poly1, poly2, y, poly, rec, init, Init, init3, i, InitT, j, g;

    userinfo(5,'MS', "Finding Recurrence for P^{f g} given P^f and P^g");

    poly1 := convert_poly(recur1, f1, x1, init1);
    poly2 := convert_poly(recur2, f2, x2, init2);

    y := `egf/result/variablename/y`;
    poly := resultant(subs(x1=x1-y,poly1),subs(x2=y,poly2),y);
    poly := expand(poly);
    poly := radnormal(poly);
    poly := expand(poly);
    rec := convert_rec(poly, f1, x1);

    userinfo(5,'MS', "Finding inital values for P^{f g} given P^f and P^g");
    g := `egf/result/procname/g`:
    init3 := subs(f2=g, init2);
    Init := [];
    for i from 0 to min(nops(init1),nops(init2))-1 do
        InitT :=  add(f1(j)*g(i-j)*binomial(i,j),j=0..i):
        Init := [op(Init), f1(i) = expand(subs([op(init1), op(init3)], InitT))];
    od;
    init := (expand(radnormal(Init)));

    RETURN(clean(rec, f1, x1, init));
end:

# egf/strip
#    Remove extrenous zeros from e.g.f.
# Input: e.g.f.,
# Output: e.g.f.,
# Reference: Appendix A.2.2.
`egf/strip` := proc(rec, f, x, init, m, q)
    local Init, i;

    Init := NULL;
    for i in init do
        if (op([1,1], i) = q) mod m then
            Init := Init, i;
        fi;
    od;

    Init := [Init];
    RETURN(rec, f, x, Init);
end:

savelib(`egf/ms`, `egf/ms.m`);
savelib(`egf/ms/result`, `egf/ms/result.m`);
savelib(`egf/ms/rec`, `egf/ms/rec.m`);
savelib(`egf/ms/rec/multi`, `egf/ms/rec/multi.m`);
savelib(`egf/ms/linalg`, `egf/ms/linalg.m`);
savelib(`egf/ms/compress`, `egf/ms/compress.m`);
savelib(`egf/ms/linalg`, `egf/ms/linalg.m`);
savelib(`egf/ms/naive`, `egf/ms/naive.m`);
savelib(`egf/clean`, `egf/clean.m`);
savelib(`egf/strip`, `egf/strip.m`);
savelib(`egf/makeproc`, `egf/makeproc.m`);
savelib(`egf/makeproc2`, `egf/makeproc2.m`);
savelib(`egf/scale`, `egf/scale.m`);
savelib(`egf/compress`, `egf/compress.m`);
savelib(`egf/uncompress`, `egf/uncompress.m`);
savelib(`egf/init`, `egf/init.m`);
savelib(`egf/result`, `egf/result.m`);
```

E.5 Denominator.

File name: Bottom.

```
## Notation:
## m.s.   = multisection
## r.p.e. = rational poly-exponential function
## p.e.   = poly-exponential function
## e.g.f. = exponential generating function

macro(`Fac` = readlib('`bottom/ms/linalg/fft2/factorial`'),
      ifactors = readlib(ifactors),
      `Expand` = readlib(`bottom/ms/linalg/fft2/expand`),
      `egf/clean` = readlib(`egf/clean`),
      `egf/init` = readlib(`egf/init`),
      `egf/result` = readlib(`egf/result`),
      `egf/ms/rec/multi` = readlib(`egf/ms/rec/multi`),
      `egf/scale` = readlib(`egf/scale`));

# bottom/ms/naive
#     M.s. the bottom of a r.p.e. using the naive method
#     of using the product as given in Lemma 3.1.
# Input: p.e., m
# Output: e.g.f.
# Reference: Lemma 3.1.
# Description Appendix A.5.1.

`bottom/ms/naive` := proc(pe, f, var, m)
    local omega, egf, pe_m, k;

    userinfo(1, 'MS', "Using naive method to find exponential generating".
        " function");
    omega := (k,m) -> exp(2*Pi*I*k/m);

    # Ref Lemma 3.1.
    pe_m := (product(subs(var=var*omega(k,m),pe),k=1..m));
    egf := convert_egf(pe_m, f, var);
    RETURN(`egf/clean`(egf));
end:

# bottom/ms/linalg/fft
#     M.s. the bottom of a r.p.e. using a combination of
#     linear algebra and the \fft\ method of fast
#     polynomial multiplication. N is the size of
#     the recurrence (less gaps). So (exp(x)-1), x, 8
#     would use an N of 10.
# Input: p.e., m, (optional) N
# Output: e.g.f.
# Reference: Subsection 5.2.1
# Description Appendix A.5.2.
`bottom/ms/linalg/fft` := proc(pe, f, var, m, N)
    local p, d, Poly, poly, FF, initial, i, rec, C, M, Zero;

    # Ref Lemma 2.5
    if nargs = 5 then
        M := N*m;
    else
        M := `pe/metric/P`(pe,var)^m+(m-1)*(`pe/metric/d`(pe,var)+1);
    fi;

    userinfo(1, 'MS', "Finding polynomial approximation for the
        poly-exponential function of degree", 2*M+1);
    Poly := (2*M)!*(convert(taylor(pe,var=0,2*M+1),polynom));

    d := 1;

    # Ref: Subsection 5.2.1.
    userinfo(1, 'MS', "Using fft to find a poly approx for the ".
        "bottom for the given poly-exponential function");
    while m <> d do
        p := ifactors(m/d)[2][1][1];
        d := d * p;

        userinfo(2, 'MS', "Dealing with primative", d, "roots of unity");
        for i from 0 to p-1 do
            poly[i] := subs(var=var*(-1)^(2*i/d),Poly);
        od;
        Poly := poly[0];
        for i from 1 to p-1 do
            if M > 250 then
                Poly := Expand(Poly, poly[i], var, m, 2*M+1)/(2*M)!;
            else
                Poly := convert(series(expand(Poly* poly[i]),var,2*M+1),
                    polynom)/(2*M)!: fi;
        od;
        Poly := radnormal(Poly);
    od;

    Poly := Poly /(2*M)!;

    Zero := `pe/metric/d`(pe, var)+1;
    for i from m*ceil(Zero*p/m) to 2*M by m do
        C[i/m-ceil(Zero*p/m)+1] := coeff(Poly,var,i)*i!;
    od;

    userinfo(1, 'MS', "Using linear algebra to determine recurrence");
    # Ref: Section 4.3.
    rec := `recurrence/solve/linalg`(C, f, var, m);

    FF := proc(i, m, q, Poly)
        if (i = q) mod m then
            RETURN(coeff(Poly,var,i)*i!);
        else
            RETURN(0);
        fi;
    end;

    initial := [seq(f(i)=FF(i, m, 0, Poly), i=0..M - 1)];

    RETURN(`egf/clean`(rec, f, var, initial));
end:

# bottom/ms/linalg/sym
#     M.s. the bottom of a r.p.e. using a combination of
#     linear algebra and symbolic differentiation.
#     N is the size of the recurrence (less gaps).
#     So (exp(x)-1), x, 8 would use an N of 10.
# Input: p.e., m, (optional) N
# Output: e.g.f.
# Reference: Section 4.4.
# Description Appendix A.5.3.

`bottom/ms/linalg/sym` := proc(pe, f, var, m, N)
    local i, egf, Pe, NN;

    # Ref Lemma 3.1.
    userinfo(1,'MS',"Taking the product of the poly-exponential function".
        " symbolically");
    Pe := expand(product(subs(var=var*exp(2*Pi*I*i/m), pe),i=1..m));

    if nargs = 5 then
        egf := `pe/ms/linalg/sym`(Pe, f, var, m, 0, N);
    else
        NN := `pe/metric/P`(Pe, var);
        egf := `pe/ms/linalg/sym`(Pe, f, var, m, 0, ceil(NN/m));
    fi;
    RETURN(`egf/clean`(egf));
end:

# bottom/ms/result
#     M.s. the bottom of a r.p.e. using a resultant
#     methods on the recurrence polynomial
#     This will give a valid recurrence relation,
#     although not necessarily minimal
# Input: p.e., m
# Output: e.g.f.
# Reference: Section 5.1.
# Description Appendix A.5.4.
`bottom/ms/result` := proc(pe, f, var, m)
    local Recur, recur, p, d, init, i, Init, size, egf, degr;

    d := 1;

    userinfo(1, 'MS', "Finding recurrision of the poly-exponential function");

    egf := [convert_egf(pe, f, var)];
    Recur := egf[1];
    Init := egf[4];


    # Ref: Section 5.1.
    userinfo(1, 'MS', "Using resultant to find a recursion for the ".
                      "bottom for the given poly-exponential function");

    while m <> d do
        p := ifactors(m/d)[2][1][1];
        d := d * p;

        userinfo(2, 'MS', "Dealing with primative", d, "roots of unity");
        size := `egf/metric/P`(Recur, f, var, Init);
        Init := `egf/init`(Recur, f, var, Init, size * m , 1, 0);

        for i from 0 to p-1 do
            recur[i] := `egf/scale`(Recur, f, var, Init, (-1)^(2*i/d));
            init[i] := recur[i][4];
            recur[i] := recur[i][1];
        od;
        Recur := recur[0];
        Init := init[0];
        for i from 1 to p-1 do
            Recur := `egf/result`(Recur, f, var, Init,
                    recur[i], f, var, init[i]);
            Init := Recur[4];
            Recur := Recur[1];
            userinfo(3,'MS',`Recur & Init are`, Recur, Init, i);
        od;
    od;
    size := `egf/metric/P`(Recur, f, var, Init);
    Init := `egf/init`(Recur, f, var, Init, size, 1, 0);
    egf := Recur, f, var, Init;

    RETURN(`egf/clean`(egf));
end:

# bottom/ms/linalg/fft2/factorial
#     This will compute the factorial of a value in a recurrrse manner.
#     It will compute this faster than the kernel level factorial in
#     maple, (which is a major bug in maple).
#     To do this, it will store every 100th value, as computed, (so
#     1% of the information calculated is remember, we don't want much
#     more than this for memory reasons.)
#     It will act recurrsively, with jumps of either 1 or 10, as required.
# Input: n
# Output: n!
`bottom/ms/linalg/fft2/factorial` := proc(n)
    option system;
    local A;
    if n < 100 then RETURN (n!) elif (n = 0) mod 10 then
        A := ((n^10-45*n^9+870*n^8-9450*n^7+63273*n^6-269325*n^5+
            723680*n^4-1172700*n^3+1026576*n^2-362880*n)*`procname`(n-10));
        if (n=0) mod 100 then
            `procname`(n) := A;
        fi:
        RETURN(A);
    else
        RETURN(`procname`(n-1)*n);
    fi;
end:

# bottom/ms/linalg/fft2
#     M.s. the bottom of a r.p.e. using a combination of
#     linear algebra and the \fft\ method of fast
#     polynomial multiplication. After the multiplication
#     to get $\prod f(x \omega_m^-d i})$, we use linalg
#     to determine the new recurrence, and then recompute
#     the new polynomial to the required length.
#     This will cut down on the initial polynomial size.
# Input: p.e., m
# Output: e.g.f.
# Reference: Subsection 5.2.2.
# Description Appendix A.5.2.
`bottom/ms/linalg/fft2` := proc(pe, f, var, m, Factors, Sym, Deg)
    local p, d, Poly, poly, i, rec, C, M, T, egf, size, Zero, MM, MMM, Poly2,
        deg, sym, sym2, fact;

    egf := convert_egf(pe, f, var): size := `egf/metric/P`(egf);

    if nargs >= 6 then
        sym := Sym;
    else
        sym := 1;
    fi;
```

```
if nargs >= 7 then
    deg := copy(Deg);
fi;

if nargs >= 5 then
    fact := Factors;
else
    fact := ifactors(m);
    fact := fact[2];
    fact := map(x->(x[1]$(x[2])),fact);
fi;

userinfo(1, 'MS', "Using fft to find a poly approx for the ".
    "bottom for the given poly-exponential function");

# Ref: Subsection 5.2.2.
d := 1;
sym2 := 1;
while m <> d do
    p := fact[1];
    fact := fact[2..-1];
    d := d * p;

    if (sym = 0) mod p then
        userinfo(2, 'MS', "Skipping primative ". d. "th roots of unity".
            " cause of symmetry");
        sym := sym / p;
        sym2 := sym2 * p;
        next;
    fi;

    userinfo(2, 'MS', "Dealing with primative ". d. "th roots of unity");

    # Ref: Lemma 2.5.
    if nargs >= 7 then
        M := deg[1];
        deg := deg[2..-1];
    else
        M := (size^p) + p*(`egf/metric/d`(egf)+1);
    fi;
    T := `egf/makeproc`(egf);

    userinfo(3, 'MS', "Determining polynomial to degree", 2*M,
        "Every", d/p, "term is present");


    Poly := 0: MM := Fac(2*M):
    MMM := MM:
    for i from 0 to floor(2*M/d*sym2*p) do
        Poly := Poly + T(d*i/p/sym2)*var^(d*i/p/sym2)*MM;
        MM := MM/product(d/p/sym2*i+j,j=1..d/p/sym2);
        if (i = 0) mod 10 then
            userinfo(6, 'MS', "Determined ", i*d/p/sym2, "term.");
        fi;
    od:

    userinfo(5, 'MS', "Scaling polynomials");
    # for i from 0 to p-1 do
    # poly[i] := subs(var=var*(-1)^(2*i/d),Poly);
    # od;

    userinfo(5, 'MS', "Multiplying the polynomials together");
    Poly2 := subs(var=var*(-1)^(2*(p-1)/d),Poly):
    for i from p-2 to 0 by -1 do

        userinfo(5, 'MS', "Scaling polynomials");
        poly := subs(var=var*(-1)^(2*i/d),Poly);

        if M > 250 then
            Poly2 := Expand(Poly2, poly, var, m*d/p, 2*M+1)/MMM;
        else
            Poly2 := convert(series(expand(Poly2 * poly),var,2*M+1),
                polynom)/MMM;
        fi;

        userinfo(6, 'MS', "Multiplied the ".i."th polynomial in");
        Poly2 := radnormal(Poly2):
        userinfo(6, 'MS', "Normalized the polynomial");
    od;
    Poly := Poly2/MMM;

    Poly2 := `Poly2`:
    Poly := radnormal(Poly);

    userinfo(3, 'MS', "Determining coefficents from polynomial");
    Zero := `egf/metric/d`(egf)+1;
    for i from d/sym2*ceil(Zero*p/d) to 2*M by d/sym2 do
        C[i/d*sym2-ceil(Zero*p/d)+1] := coeff(Poly,var,i)*Fac(i);
    od;

    userinfo(3, 'MS', "Determining recurrence for polynomial with linalg");
    rec := `recurrence/solve/linalg`(C, f, var, d/sym2);#, "toeplitz");


    egf := rec, f, var, [seq(f(i)=coeff(Poly,var,i)*Fac(i), i=0..M - 1)];
    size := `egf/metric/P`(egf): C := 'C';
od;

RETURN(`egf/clean`(rec, f, var,
    [seq(f(i)=coeff(Poly,var,i)*Fac(i), i=0..size - 1)]));
end:

# bottom/ms/factor
#     M.s. the bottom using any method mentioned, but factors out
#     any polynomials first, which it returns as a last argument
# Input: p.e., m, method[methodarg]
# Output: e.g.f., scale
`bottom/ms/factor` := proc(pe, f, var, m, opt)
    local i, method, methodarg, egf, Pe, Poly, j;

    userinfo(1, 'MS', "Removing common polynomials before determining".
        " exponential generating function");
    if nargs = 5 then
        if type(opt, indexed) then
            method := `bottom/ms/`.(op(0, opt));
            methodarg := op(1, opt);
        else
            method := `bottom/ms/`.opt;
        fi;
    else
        method := `bottom/ms/linalg/fft2`;
    fi;

    Pe := factor(pe);
    if type(Pe,`*`) then
        Poly := select(x->(type(x,polynom(anything,var))),[op(Pe)]);
        Pe := select(x->(not type(x,polynom(anything,var))),[op(Pe)]);
        Poly := mul(j,j=Poly);
        Pe := mul(j,j=Pe);
    else
        if type(Pe,polynom(anything,var)) then
            Poly := Pe;
        else
            Poly := 1;
        fi;
    fi;

    if assigned(methodarg) then
        egf := method(Pe,f,var,m,methodarg);
    else
        egf := method(Pe,f,var,m);
    fi;

    Poly := `egf/ms/rec/multi`(Poly,var,m,1):
    RETURN(`egf/clean`(egf), Poly);
end:
```

```
# bottom/ms
#     M.s. the bottom of the r.p.e. with a p.e. bottom by m
# Input: p.e., m, method[methodarg]
# Output: e.g.f.
`bottom/ms` := proc(pe, f, var, m, opt)
    local i, method, methodarg, egf;

    userinfo(1, 'MS', "Dealing with the bottom of the r.p.e.");
    if nargs = 5 then
        if type(opt, indexed) then
            method := `bottom/ms/`.(op(0, opt));
            methodarg := op(1, opt);
        else
            method := `bottom/ms/`.opt;
        fi;
    else
        method := `bottom/ms/linalg/fft2`;
    fi;

    if assigned(methodarg) then
        egf := method(pe,f,var,m,methodarg);
    else
        egf := method(pe,f,var,m);
    fi;

    RETURN(`egf/clean`(egf));
end:

# bottom/ms/linalg/fft2/expand
# Expands the product of two polynomials. Attempts to use
# less memory than the maple kernal equivalent.
# It will look at the different components of the polynomial,
# where the degree falls into different residuals modulo omega.

# Input: poly1, poly2, var, omega, cutoff
# Output: poly1*poly2
`bottom/ms/linalg/fft2/expand` := proc(poly1, poly2, var, omega, cutoff)
    local p1, p2, y, i, j, p, Poly, A, T;

    for i from 0 to omega-1 do
        p1[i mod omega] := 0:
        p2[i mod omega] := 0: i
    od:

    for i from 0 to omega - 1 do
        userinfo(6,'MS',"Got information for omega ".i.".");
        p1[i mod omega] := add(var^(omega*j + i)*coeff(poly1, var, omega*j+i),
            j= 0...ceil(cutoff/omega)+1);
        p2[i mod omega] := add(var^(omega*j + i)*coeff(poly2, var, omega*j+i),
            j=0...ceil(cutoff/omega)+1);
    od;

    for i from 0 to omega - 1 do
        p[i] := 0:
    od:

    for i from 0 to omega - 1 do
        for j from 0 to omega - 1 do
            userinfo(6,'MS',"Dealing with p1[".i."], and p2[".j."]");
            if nargs = 5 then
                p[(i+j) mod omega] :=
                    p[(i+j) mod omega] +
                    convert(series(expand(p1[i]*p2[j]),var,cutoff+1),polynom);
            else
                 p[(i+j) mod omega] :=
                    p[(i+j) mod omega] + expand(p1[i]*p2[j]);
            fi;
        od;
    od;

    userinfo(6,'MS',"Adding back together"):
    Poly := add(p[i],i=0..omega-1);

    RETURN(Poly):
end:

#libname := libname[3], libname[1..2]:
savelib(`bottom/ms/naive`, `bottom/ms/naive.m`);
savelib(`bottom/ms/linalg/fft`, `bottom/ms/linalg/fft.m`);
savelib(`bottom/ms/linalg/sym`, `bottom/ms/linalg/sym.m`);
savelib(`bottom/ms/result`, `bottom/ms/result.m`);
savelib(`bottom/ms/linalg/fft2`, `bottom/ms/linalg/fft2.m`);
savelib(`bottom/ms/linalg/fft2/expand`, `bottom/ms/linalg/fft2/expand.m`);
savelib(`bottom/ms/factor`,`bottom/ms/factor.m`);
savelib(`bottom/ms`,`bottom/ms.m`);
savelib(`bottom/ms/linalg/fft2/factorial`,`bottom/ms/linalg/fft2/factorial.m`);
```

E.6 Numerator.

File name: Top.

```
## Notation:
## m.s.   = multisection
## r.p.e. = rational poly-exponential function
## p.e.   = poly-exponential function
## e.g.f. = exponential generating function

macro(`egf/clean` = readlib(`egf/clean`),
      `egf/result` = readlib(`egf/result`),
      `egf/scale` = readlib(`egf/scale`),
      `egf/init` = readlib(`egf/init`),
      `egf/ms/rec/multi` = readlib(`egf/ms/rec/multi`));

# top/ms/naive
#     M.s. the top of the r.p.e. using the naive method.
# Input: p.e. (top), p.e. (bottom), m, q
# Output: e.g.f.
# References: Lemma 3.1.
#             Appendix A.6.1.
`top/ms/naive` := proc(top, bot, f, var, m, q)
    local omega, egf, pe_2, k;

    userinfo(1,'MS',"Using naive method to find exponential ".
        "generating function");

    # Ref Lemma 3.1.
    pe_2 := (top*product(subs(var=var*(-1)^(2*k/m),bot),k=1..m-1));
    egf := `pe/ms/naive`(pe_2, f, var, m, q);
    RETURN(`egf/clean`(egf));
end:

# top/ms/linalg/fft
#     M.s. the top of a r.p.e. using a combination of
#     linear algebra and the \fft\ method of fast
#     polynomial multiplication.  N is the size of
#     the recurrence (less gaps).  So (exp(x)-1), x, x, 8
#     would use an N of 20.
# Input: p.e. (top), p.e. (bottom), m, (optional) N
# Output: e.g.f.
# Reference: Section 5.2.
#            Appendix A.6.2.
`top/ms/linalg/fft` := proc(top, bot, f, var, m, q, N)

    local Poly, poly, FF, initial, i, rec, C, M, Zero;

    # Ref Lemma 3.6.
    Zero := `pe/metric/d`(top,var)+`pe/metric/d`(bot,var)*(m-1)+1;
    if nargs = 7 then
        M := N*m;
    else
        M := m*(`pe/metric/P`(top,var)+1)*(`pe/metric/P`(bot,var)+1)^(m-1)+Zero;
    fi;

    userinfo(1, 'MS', "Finding polynomial approximation for the pe of size",
        2*M+Zero);
    Poly := (2*M+Zero)!*(convert(taylor(bot,var=0,2*M+Zero+1),polynom));

    poly := (2*M+Zero)!*convert(taylor(top,var=0,2*M+Zero+1),polynom);

    # Ref: Section 5.2.
    userinfo(1, 'MS', "Using fft to find a poly approx for the ".
                      "top for the given pe");
    for i from 1 to m-1 do
        poly := convert(series(expand(poly *
            subs(var=var*(-1)^(2*i/m),Poly)),var,2*M+Zero),polynom)/(2*M+Zero)!;
#        poly := convert(series(expand(poly *
#            subs(var=var*exp(2*Pi*I*i/m),Poly)),var,2*M), polynom)/(2*M)!;
    od;

    poly := radnormal(poly / (2*M+Zero)!);

    for i from q+m*ceil(Zero/m) to 2*M by m do
        C[i/m-ceil(Zero/m)-q/m+1] := coeff(poly,var,i)*i!;
    od;
#    for i from Zero to 2*M by m do
#        C[i-Zero+1] := coeff(poly,var,i)*i!;
#    od;

    userinfo(1, 'MS', "Using linear algebra to determine recurrence");
    rec := `recurrence/solve/linalg`(C, f, var, m);

    FF := proc(i, m, q, poly)
        if (i = q) mod m then
            RETURN(coeff(poly,var,i)*i!)
        else
            RETURN(0);
        fi;
    end;

    initial := [seq(f(i)=FF(i, m, q, poly), i=0..M - 1 + q + Zero)];

    RETURN(`egf/clean`(rec, f, var, initial));
end:


# top/ms/linalg/sym
#     M.s. the top of a r.p.e. using a combination of
#     linear algebra and symbolic differentiation.
#     N is the size of the recurrence (less gaps).
#     So (exp(x)-1), x, x, 8 would use an N of 20.
# Input: p.e., m, (optional) N
# Output: e.g.f.
# Reference: Section 4.3.
#            Appendix A.6.3.
`top/ms/linalg/sym` := proc(top, bot, f, var, m, q, N)
    local i, egf, Pe;

    # Ref: Lemma 3.1.
    userinfo(1,'MS',"Taking the product of the poly-".
        "exponential functions symbolically");
    Pe := expand(product(subs(var=var*exp(2*Pi*I*i/m), bot),i=1..m-1)*top);
#    Pe := expand(product(subs(var=var*(-1)^(2*i/m), bot),i=1..m-1)*top);

    if nargs = 7 then
        egf := `pe/ms/linalg/sym`(Pe, f, var, m, q, N);
    else
        egf := `pe/ms/linalg/sym`(Pe, f, var, m, q);
    fi;
    RETURN(`egf/clean`(egf));
end:

# top/ms/result
#     M.s. the top of a r.p.e. using a resultant
#     methods on the recurrence polynomial
#     This will give a valid recurrence relation,
#     although not necessarily minimal
# Input: p.e. (top), p.e. (bottom), m
# Output: e.g.f.
# Reference: Section 5.1.
#           Appendix 6.6.
`top/ms/result` := proc(top, bot, f, var, m, q)

    local RecurB, recur,
        p, d, poly, FF, init, i, rec, C, InitB, size, egf, egfB,
        recurB, initB, Size;

    d := 1;

    userinfo(1, 'MS', "Finding recurrision of the top and bottom");
    egfB := [convert_egf(bot, f, var)];
    egf := [convert_egf(top, f, var)];
    recur := egf[1];
    init := egf[4];
    RecurB := egfB[1];
    InitB := egfB[4];
    Size := `egf/metric/P`(op(egfB));

    # Ref: Section 5.1.
    userinfo(1, 'MS', "Using resultant to find a recursion for the ".
                      "top for the given poly-exponential functions");
    for d from 1 to m-1 do
        size := `egf/metric/P`(recur, f, var, init) * Size;
        init := `egf/init`(recur, f, var, init, size, 1, 0);
        recurB := `egf/scale`(RecurB, f, var, InitB, (-1)^(2*d/m));
        initB := recurB[4];
        recurB := recurB[1];
        initB  := `egf/init`(recurB, f, var, initB, size, 1, 0);
        initB  := radnormal(initB);
        recur := `egf/result`(recurB, f, var, initB, recur, f, var, init);
        init := recur[4];
        recur := recur[1];
        init := map(radnormal,init);
    od;

    size := `egf/metric/P`(recur, f, var, init);
    init := `egf/init`(recur, f, var, init, size, 1, 0);

    egf := `egf/ms/rec`(recur, f, var, init, m, q);

    egf := op(radnormal([egf]));

    RETURN(`egf/clean`(egf));
end:

# top/ms/linalg/know
#     M.s. the top of a r.p.e. using a combination of
#     linear algebra and knowledge about the bottom, and actual
#     recurrence
#     N is the size of the recurrence (less gaps).
#     zero is the number of bad initial values to skip (defaults to 2)
# Input: proc (bot), proc (actual), m, N, (optional) zero
# Output: e.g.f.
# Reference: Section 5.3.
#           Appendix A.6.5.
`top/ms/linalg/know` := proc(botP, actP, f, var, m, q, N, zero, shift)
    local i, egf, Pe, Zero, j, temp, C, rec, initial, Shift;

    if nargs >= 9 then
        Shift := shift;
    else
        Shift := 0;
    fi;

    if nargs >= 8 then
        Zero := zero;
    else
        Zero := 2;
    fi;

    initial := [seq(f(i)=0,i=0..Shift-1)];
    userinfo(1, 'MS', "Determining top values");
    for i from Shift to 2 * N *m + Zero do
```

```
        j := 'j':
        if (i = q+Shift) mod m then
            temp := add(binomial(i, q+j*m)*actP(m*j+q)*botP(i-q-j*m),
                j=0..(i-q)/m);
        else
            temp := 0;
        fi;
        if (i = 0) mod 10 then
            userinfo(2, 'MS', "Determining value ".i);
        fi;
        if i > Zero and (i = q+Shift) mod m then
            C[(i-q-Shift-ceil((Zero-q-Shift+1)/m)*m)/m+1] := temp;
        fi;
        initial := [op(initial),f(i)=temp];
    od;

    userinfo(1, 'MS', "Using linear algebra to determine recurrence");
    rec := `recurrence/solve/linalg`(C, f, var, m);#, "toeplitzf");

    egf := rec, f, var, initial;

    RETURN(`egf/clean`(egf));
end:

# top/ms/factor
#     M.s. the top using any method mentioned, but factors out
#     any polynomials first, which it returns as a last argument
# Input: p.e. (top), p.e. (bot), m, q, method[methodarg]
# Output: e.g.f., scale
`top/ms/factor` := proc(top, bot, f, var, m, q, opt)
    local i, method, methodarg, egf, Pe, Poly, j, Top, PolyT, Bot,
        PolyB, T, g, B, newq;

    userinfo(1,'MS',"Removing common polynomials before determining ".
        "exponential generating function");
    if nargs = 7 then
        if type(opt, indexed) then
            method := `top/ms/`.(op(0, opt));
            methodarg := op(1, opt);
        else
            method := `top/ms/`.opt;
        fi;
    else
        method := `top/ms/linalg/fft`;
    fi;

    Top := factor(top);
    if type(Top,`*`) then
        PolyT := select(x->(type(x,polynom(anything,var))),[op(Top)]);
        PolyT := mul(j,j=PolyT);
    else
        if type(Top,polynom(anything,var)) then
            PolyT := Top;
        else
            PolyT := 1;
        fi;
    fi;

    Bot := factor(bot);

    if type(Bot,`*`) then
        PolyB := select(x->(type(x,polynom(anything,var))),[op(Bot)]);
        PolyB := mul(j,j=PolyB);
    else
        if type(Bot,polynom(anything,var)) then
            PolyB := Bot;
        else
            PolyB := 1;
        fi;
    fi;

    T := product(subs(var=var*(-1)^(2*i/m),PolyB),i=1..(m-1))*PolyT;
    T := simplify(T):
    PolyT := simplify(PolyT):
    PolyB := simplify(PolyB):
    g := T;
    for i from 1 to m-1 do
        g := gcd(g, simplify(subs(var=var*(-1)^(2*i/m), T)));
        g := simplify(g):
        if degree(g,var) = 0 then
            g := 1;
            break;
        fi;
    od;

    PolyT := gcd(PolyT, g);

    T := `egf/ms/rec/multi`(PolyB,var,m,1);
    T := gcd(T,g):

    PolyB := quo(T, simplify(g/PolyT), var):

    Bot := Bot/PolyB;
    Top := Top/PolyT;

    if type(g, `+`) then
        if nops({op(map(x->x mod m, map(degree,[op(randpoly(x))])))}) = 1 then
            newq := (q-degree(g,var)) mod m;
        else
            newq := "all";
        fi;
    else
        newq := (q-degree(g,var)) mod m;
    fi;

    if assigned(methodarg) then
        egf := method(Top,Bot,f,var,m,newq,methodarg);
    else
        egf := method(Top,Bot,f,var,m,newq);
    fi;

    RETURN(`egf/clean`(egf), g);
end:

# top/ms
#     M.s. the top of the r.p.e. by m
# Input: p.e. (top), p.e. (bot) m, method[methodarg]
# Output: e.g.f.
`top/ms` := proc(top, bot, f, var, m, q, opt)
    local i, method, methodarg, egf;

    userinfo(1, 'MS', "Dealing with the bottom of the rational ".
        "poly-exponential function");

    if nargs = 7 then
        if type(opt, indexed) then
            method := `top/ms/`.(op(0, opt));
            methodarg := op(1, opt);
        else
            method := `top/ms/`.opt;
        fi;
    else
        method := `top/ms/linalg/fft`;
    fi;

    if assigned(methodarg) then
        egf := method(top, bot, f, var, m, q, methodarg);
    else
        egf := method(top, bot, f, var, m, q);
    fi;

    RETURN(`egf/clean`(egf));
end:

# top/ms/know
#     M.s. the top of a r.p.e. using knowledge about the bottom, and actual
#     values, and the recurrence
#     N is the size of the recurrence (less gaps).
# Input: recurrence, proc (bot), proc (actual), m, N
# Output: e.g.f.
# Reference: Section 5.3.
#           Appendix A.6.6.
`top/ms/know` := proc(rec, botP, actP, f, var, m, q, N)
    local C, init, egf, i, m1, q1, j:

    C := (i, m1, q1) -> add(binomial(i, q1+j*m1)*actP(m1*j+q1)*botP(i-q1-j*m1),
                 j=0..(i-q1)/m1);

    userinfo(2, 'MS', "Getting initial values");
    init := [seq(f(i) = C(i, m, q), i = 0 .. N*m)];

    egf := `egf/clean`(rec, f, var, init);

    RETURN(egf);
end:

#libname := libname[3], libname[1..2]:
savelib(`top/ms/naive`, `top/ms/naive.m`);
savelib(`top/ms/linalg/fft`, `top/ms/linalg/fft.m`);
savelib(`top/ms/linalg/sym`, `top/ms/linalg/sym.m`);
savelib(`top/ms/result`, `top/ms/result.m`);
savelib(`top/ms/linalg/know`, `top/ms/linalg/know.m`);
savelib(`top/ms/factor`, `top/ms/factor.m`);
savelib(`top/ms`, `top/ms.m`);
savelib(`top/ms/know`, `top/ms/know.m`):
```

E.7 Linear Algebra.

File name: Linalg.

```
macro(linsolve = readlib(linalg)[linsolve],
      rDot     = readlib(`recurrence/solve/toeplitz/rdot`),
      HankelSolver = readlib(`recurrence/solve/hankel/solver`),
      Rev      = readlib(`recurrence/solve/toeplitz/rev`));

# recurrence/solve/linalg
#     Solves the recurrence relationship given the first
#     few initial values.  The recurrence relationship returned
#     will be using the function and variable given.
# Input: Value, fun, var, m
# Output: Recurrence relationship
# References: Section 4.3
`recurrence/solve/linalg` := proc(Value, fun, var, m, toe)
    local i, j, N, C, b, ans, rec;

    save Value, "Value".m."Problem";

    if true then #nargs=5 and toe = "hankel" then
        RETURN(readlib(`recurrence/solve/hankel`)(Value, fun, var, m));
    elif nargs=5 and toe = "toeplitz" then
        RETURN(readlib(`recurrence/solve/toeplitz`)(Value, fun, var, m));
    elif nargs=5 and toe = "toeplitzf" then
        RETURN(readlib(`recurrence/solve/toeplitzf`)(Value, fun, var, m));
    fi;

    userinfo(3, 'MS', "Using linear algebra to determine the recurrence");
    if type(Value,table) then
        N := floor(nops(op([1,2],Value))/2);
    elif type(Value,list) then
        N := floor(nops(Value)/2);
    fi;

    userinfo(4, 'MS', "Finding matrix of size ". N. " X ". N.".");
    C := matrix(N,N):

    for i from 1 to N do
        for j from 1 to N do
            C[i,j] := Value[i+j-1];
        od;
    od;

    userinfo(4, 'MS', "Finding vector of size ". N.".");
    b := vector([seq(Value[i+N],i=1..N)]);

    ans := linsolve(C,b);
    ans := convert(ans,list);

    i := 1;
    do
        if has(ans, _t[i]) then
            for j from 1 to N do
                if has(ans[j] , _t[i]) then
                    ans := subs(_t[i] = solve(ans[j], _t[i]),ans);
                    break;
                fi;
            od;
        else
            break;
        fi;
        i := i + 1;
    od;

    rec := fun(var) = add(ans[i]*fun(var-(N+1)*m+i*m),i=1..N);

    userinfo(5, 'MS', "Returing recurrsion"):
    RETURN(rec);
end:


`recurrence/solve/hankel` := proc(Value, fun, var, m)
    local N, H, X, i, rec;
    userinfo(3, 'MS', "Using George's methods algebra to".
        " determine the recurrence");
    if type(Value,table) then
        N := floor((nops(op([1,2],Value))-1)/2);
    elif type(Value,list) then
        N := floor((nops(Value)-1)/2);
    fi;

    H := matrix(N,N+1,[seq(seq(Value[i+j],i=1..N+1),j=1..N)]):

    userinfo(4, 'MS', "Finding matrix of size ". N. " X ". (N+1).".");
    X := HankelSolver(H):

    if abs(X[N+1,1]) <> 1 then print("Something is horribly wrong".
        " 2*N needs to be bigger than ". (2*N));
        RETURN("ERROR");
    fi;

    rec := fun(var) = add(-X[N+1,1]*X[i,1]*fun(var-(N+1)*m+i*m),i=1..N);

    userinfo(5, 'MS', "Returing recurrsion"):
    RETURN(rec);
end:

`recurrence/solve/hankel/solver` := proc(A)
    local i, z, C, F, n;

    n := linalg[rowdim](A);

    C :=series(add(A[1,i]*z^(i-1),i=1..n)+add(A[n,i]*z^(n+i-2),i=2..n+1),z,
        2*n+1);

    F := denom( convert( C, ratpoly, n-1,n ));

    matrix(n+1,1,[seq(coeff(F,z,n-i),i=0..n)]);

end:


# Examples which I ran it on just as a check:
```

```
`recurrence/solve/toeplitz/rdot` := proc(a,b)
    local i, ans, n;
    if a = 0 then RETURN(0); fi;
    n := nops(a);
    ans := 0;
    for i from 1 to nops(a) do
        ans := a[i] * b[1+n-i] + ans;
    od;
end:

`recurrence/solve/toeplitz/rev` := proc(a)
    local i, n, ans;
    if a = 0 then RETURN(0); fi;
    ans := [seq(a[nops(a)+1-i],i=1..nops(a))];
    RETURN(ans);
end:

`recurrence/solve/toeplitz` := proc(Value, fun, var, m)
    local r, s, y, f, g, delta, gamma, N, rp, sp, C, i, j, t, OldN, OldN2,
          ans, rec, Vvalue;

#    save Value, ToeplitzValue.m;

    if type(Value,table) then
        N := nops(op([1,2],Value));
        Vvalue := NULL;
        for i from 1 to N do
            Vvalue := Vvalue, Value[i];
        od;
        Vvalue := [Vvalue];
#        Vvalue := convert(Value, list);
    fi;

    N := floor(nops(Vvalue)/2);
#print("Original N", N);

    OldN2 := N;

    while Vvalue[N] = 0 do N := N-1 od;

    OldN := N:

    #B := matrix(N,N,[seq(seq(A(j-i+N-1),i=0..N-1),j=0..N-1)]);

    t[0] := Vvalue[N];

    userinfo(3, 'MS', "Using toeplitz method to determine the recurrence");
    for j from 1 to N-1 do
        userinfo(4, 'MS', "Setting up ".j."-th term of ".(N-1).".");
        r[(N-j)] := Rev(Vvalue[j .. N-1]);
        s[(N-j)] := Vvalue[N+1 .. 2*N-j];
        rp[j] := Vvalue[N-j];
        sp[j] := Vvalue[N+j];
    od:

    y[0] := 1/t[0];
    f[0] := 0;
    g[0] := 0;

    for i from 0 to N-2 do
        userinfo(4, 'MS', "Solving up ".i."-th problem of ".(N-2).".");
        gamma[i] := y[i] * rp[i+1] + rDot(f[i], r[i]);
        delta[i] := y[i] * sp[i+1] + rDot(g[i], s[i]);
        if (delta[i] * gamma[i] = 1) then
            N := i + 1;
            break;
        fi;
        y[i+1] := y[i] / (1-delta[i] * gamma[i]);
        if i = 0 then
            f[i+1] := y[i+1]/y[i] * [-gamma[i] * y[i]];
            g[i+1] := y[i+1]/y[i] * [-delta[i] * y[i]];
        else
            f[i+1] := y[i+1]/y[i] * [op(f[i] - gamma[i] * Rev(g[i])),
                -gamma[i] * y[i]];
            g[i+1] := y[i+1]/y[i] * [op(g[i] - delta[i] * Rev(f[i])),
                -delta[i] * y[i]];
        fi;
    od:

    C := matrix(N,N);
    C[1,1] := y[N-1];
    for i from 1 to N-1 do
        C[1,i+1] := f[N-1][i];
        C[i+1,1] := g[N-1][i];
    od:
    for i from 1 to (N-2) do
        C[N,i+1] := g[N-1][N-1-i];
        C[i+1,N] := f[N-1][N-1-i];
    od:
    C[N,N] := y[N-1]:
#    print(C);
    for i from 1 to N-2 do
        for j from 1 to N-2 do
            userinfo(4, 'MS', "Finding value for (".i.",".j.")-th entry");
            C[i+1,j+1] := C[i,j] + 1/C[1,1] * (C[i+1,1]*C[1,j+1] -
                C[1,N-i+1] * C[N-j+1,1]);
        od;
    od;

    i := 'i';
#    print(matrix(N,1,[seq(Vvalue[OldN+i],i=1..N)]));
    ans := evalm(C &* matrix(N,1,[seq(Vvalue[OldN2+i],i=1..N)]));
#print("N, OldN, OldN2", N, OldN, OldN2, "ans", ans);

    rec := fun(var) = add(ans[N+1-i,1]*fun(var-((OldN2-OldN)+N+1)*m+i*m),
        i=1..N);
    RETURN(rec);
end:

`recurrence/solve/toeplitzf` := proc(Value, fun, var, m)
    local r, s, y, f, g, delta, gamma, N, rp, sp, C, i, j, t, OldN,
          ans, rec, Vvalue;

#    save Value, ToeplitzfValue.m;

    if type(Value,table) then
        N := nops(op([1,2],Value));
        Vvalue := NULL;
        for i from 1 to N do
            Vvalue := Vvalue, Value[i];
        od;
        Vvalue := [Vvalue];
    fi;
    N := floor(nops(Vvalue)/2);

    OldN := N;

    while Vvalue[N] = 0 do N := N-1 od;


    Digits := ceil(sqrt(N)*max(op(map(x->log[10](abs(x)), Vvalue))));

    Vvalue := map(evalf, Vvalue);


    #B := matrix(N,N,[seq(seq(A(j-i+N-1),i=0..N-1),j=0..N-1)]);

    t[0] := Vvalue[N];

    userinfo(3, 'MS', "Using toeplitz method to determine the recurrence,".
        " with ".Digits." digits accuracy.");
    for j from 1 to N-1 do
        userinfo(4, 'MS', "Setting up ".j."-th term of ".(N-1).".");
        r[(N-j)] := Rev(Vvalue[j .. N-1]);
        s[(N-j)] := Vvalue[N+1 .. 2*N-j];
        rp[j] := Vvalue[N-j];
        sp[j] := Vvalue[N+j];
    od:
```

```
    y[0] := 1/t[0];
    f[0] := 0;
    g[0] := 0;

    for i from 0 to N-2 do
        userinfo(4, 'MS', "Solving up ".i."-th problem of ".(N-2).".");
        gamma[i] := y[i] * rp[i+1] + rDot(f[i], r[i]);
        delta[i] := y[i] * sp[i+1] + rDot(g[i], s[i]);
#print(evalf(delta[i]*gamma[i], 100));
        if (evalf(delta[i] * gamma[i], ceil(Digits/sqrt(N))) = 1.0) then
            N := i + 1;
            break;
        fi;
        y[i+1] := y[i] / (1-delta[i] * gamma[i]);
        if i = 0 then
            f[i+1] := y[i+1]/y[i] * [-gamma[i] * y[i]];
            g[i+1] := y[i+1]/y[i] * [-delta[i] * y[i]];
        else
            f[i+1] := y[i+1]/y[i] * [op(f[i] - gamma[i] * Rev(g[i])),
                -gamma[i] * y[i]];
            g[i+1] := y[i+1]/y[i] * [op(g[i] - delta[i] * Rev(f[i])),
                -delta[i] * y[i]];
        fi;
    od:

    C := matrix(N,N);
    C[1,1] := y[N-1];
    for i from 1 to N-1 do
        C[1,i+1] := f[N-1][i];
        C[i+1,1] := g[N-1][i];
    od:
    for i from 1 to (N-2) do
        C[N,i+1] := g[N-1][N-1-i];
        C[i+1,N] := f[N-1][N-1-i];
    od:
    C[N,N] := y[N-1]:
#    print(C);
    for i from 1 to N-2 do
        for j from 1 to N-2 do
            userinfo(4, 'MS', "Finding value for (".i.",".j.")-th entry");
            C[i+1,j+1] := C[i,j] + 1/C[1,1] * (C[i+1,1]*C[1,j+1] -
                C[1,N-i+1] * C[N-j+1,1]);
        od;
    od;

    i := 'i';
#    print(matrix(N,1,[seq(Vvalue[OldN+i],i=1..N)]));
    ans := evalm(C &* matrix(N,1,[seq(Vvalue[OldN+i],i=1..N)]));

#print(ans);
    ans := map(round,ans);
#print(ans);

    rec := fun(var) = add(ans[N+1-i,1]*fun(var-(N+1)*m+i*m),i=1..N);
    RETURN(rec);
end:

savelib(`recurrence/solve/linalg`, `recurrence/solve/linalg.m`);
savelib(`recurrence/solve/toeplitz/rev`, `recurrence/solve/toeplitz/rev.m`);
savelib(`recurrence/solve/toeplitz/rdot`, `recurrence/solve/toeplitz/rdot.m`);
savelib(`recurrence/solve/toeplitz`, `recurrence/solve/toeplitz.m`);
savelib(`recurrence/solve/hankel`, `recurrence/solve/hankel.m`);
savelib(`recurrence/solve/hankel/solver`, `recurrence/solve/hankel/solver.m`);
savelib(`recurrence/solve/toeplitzf`, `recurrence/solve/toeplitzf.m`);
```

E.8 Performing the calculations.

File name: Normal.

```
# calcul/normal
#     Perform the calculation using normal methods
# Input: Recurrence
# Output: Values
% Refence: Theorem 3.1.
`calcul/normal` := proc(Largest, Top, Bot, m, q, feq, File, Info)
    local i, B, info, Value, j, s, work;

    if nargs = 8 then
        B := copy(Info);
        for i from q to Largest by m do
            if has(B[i] , B) then
                work := i;
                break;
            fi;
        od;
    else
        work := q:
    fi;

    for i from 0 to infinity do
        if Bot(i) <> 0 then
            s := i;
            break;
        fi;
    od;

    for i from work to Largest by m do

        if not has(B[i] , B) then
            userinfo(3, 'MS', "Knew the ". i. "th value already.");
            next;
        fi;

        Value := Top(i+s);

        userinfo(2, 'MS', "Working on problem", i);
        for j from q to i-m by m do
            Value := Value - Bot(s+i-j)*B[j]*binomial(i+s,j);
        od;

        Value := Value / binomial(i+s,s)/Bot(s);

        userinfo(3, 'MS', "Determined ". i. "th value.");
        B[i] := Value;

        if nargs >= 7 then
            if (i = 0) mod feq then
                save B, File.i.`.m`;
            fi;
        fi;
    od;

    RETURN(copy(B));
end:

#libname := libname[3], libname[1..2]:
savelib(`calcul/normal`, `calcul/normal.m`);
```

File name: Multi.

```
macro(binomial = readlib(binomial),
      readpipe = readlib(`calcul/readpipe`),
      writepipe = readlib(`calcul/writepipe`));
```

```
# calcul/balanced/worker
#     The slave that does all the work
# Input: Recurrences
# Output: NOTHING
# Reads: Values of other calculations.
# Writes: Value to calculations performed
# Reference: Section 6.2.
‘calcul/balanced/worker‘ :=
        proc(Largest, N, work, ReadPipe, WritePipe, Top, Bot, m, q, Info)
    local i, B, info, Value, j, s, start, tt;

    tt := time():
    B := copy(Info);

    for i from work to Largest by m*N do
        if has(B[i], B) then
            start := i;
            break;
        fi;
    od;

    for i from 0 to infinity do
        if Bot(i) <> 0 then
            s := i;
            break;
        fi;
    od;

    for i from start to Largest by N*m do

        Value := Top(i+s);
        userinfo(2, ’MS’, "Slave", work, "working on problem", i);

        for j from q to max(q-m,i - N*m) by m do
            Value := Value - Bot(s+i-j)*B[j]*binomial(i+s,j);
        od;

        for j from 0 to min(i-m, m*N-2*m) by m do
            userinfo(3, ’MS’, "Slave", work, "getting needed info from Master");
            info := NULL:
            while info = NULL do
                info := readpipe(ReadPipe[work]);
            od;
            B[info[1]] := info[2];
        od;

        userinfo(3, ’MS’, "Slave", work, "finishing calculation");
        for j from max(q,i - N*m+m) to i-m by m do
            Value := Value - Bot(s+i-j)*B[j]*binomial(i+s,j);
        od;

        Value := Value / binomial(i+s,s)/Bot(s);

        userinfo(3, ’MS’, "Slave", work, "Reporting to Master");
        writepipe(WritePipe[work],[i,Value]);
        B[i] := Value;
    od;

    print("Slave ".work." took",(time() - tt), "seconds."):
    RETURN();
end:

# calcul/balanced
#     The form of communication between the workers.
# Input: Recurrences
# Output: Values
# Reads: Values of calculations.
# Writes: Value to calculations.
‘calcul/balanced‘ := proc(N, Largest, Top, Bot, m, q, feq, File, Info)
    local Slaves, Master, i, j, pid, work, info, l, B, start, i2, k;

    if nargs = 9 then
        B := copy(Info);
        for i from q to Largest by m do
            if has(B[i], B) then
                start := i;
                break;
            fi;
        od;
    else
        start := q;
    fi;

    work := q;
    for i from q to (N-1)*m+q by m do
        Slaves[i] := pipe();
        Master[i] := pipe();
    od;

    for i from 1 to N do

        pid := fork();
        if pid = 0 then # Slaves
            userinfo(1, ’MS’, "Starting up slave", work);
            readlib(‘calcul/balanced/worker‘)
                    (Largest, N, work, Slaves, Master, Top, Bot, m, q, B);
            system("sleep 1");

            userinfo(1, ’MS’, "Stopping slave", work);
            quit;
        elif i = N then # Master
            if start <> q then
                k := 1;
                 i := start mod N*m;
#                for i from (start mod N*m) to
#                        (start mod N*m) + (N-1)*m by m do
                    for j from i - (N-1)*m to i - m*k by m do
                        userinfo(3, ’MS’, "Sending info to slave", i);

#                        info := convert([j,B[j]],string);
                        writepipe(Slaves[(i mod N*m)],[j, B[j]]);
                    od;
#                    k := k + 1;
#                od;
            fi;

            for j from start to Largest by m do

                ## Get the info from the slaves.
                userinfo(3, ’MS’, "Getting information from slave",
                    (j) mod N*m );
                info := NULL;
                while info = NULL do
                    info := readpipe(Master[(j) mod N*m ]);
                od;
                B[info[1]] := info[2];
#                info := convert(info,string);

                # Send info to next slaves.
                if (j+m <= Largest) then
                    for i2 from (j-(N-2)*m) to j by m do
                        if i2 < 0 then next; fi ;
                        userinfo(3, ’MS’, "Sending info to slave",(j+m)
                                mod N*m);
#                        info := convert([i2, B[i2]],string);
                        writepipe(Slaves[(j+m) mod N*m],[i2, B[i2]]);
                    od;
                fi;

                if nargs >= 7 and ((j = 0) mod feq) then
                    userinfo(3, ’MS’, "Saving results so far");
                    save B, File.j.‘.m‘;
                fi;

            od;
        fi;

        work := work + m;
    od;

    ## Wait for all the slaves to finish
    for i from 1 to N do
        wait();
    od;

    for i from q to (m-1)*N+q by m do
        close(Slaves[i][1]);
        close(Slaves[i][2]);
        close(Master[i][1]);
        close(Master[i][2]);
    od;

    RETURN(copy(B));
end:

savelib(`calcul/balanced/worker`, `calcul/balanced/worker.m`);
savelib(`calcul/balanced`, `calcul/balanced.m`);
```

File name: Multi2.

```
macro(binomial  = readlib(binomial),
      ceil      = readlib(ceil),
      frac      = readlib(frac),
      printf   = readlib(printf),
      readpipe  = readlib(`calcul/readpipe`),
      writepipe = readlib(`calcul/writepipe`),
      readfile  = readlib(`calcul/readfile`),
      writefile = readlib(`calcul/writefile`));

# calcul/readpipe
#     Performs the reading of information from pipe
# Input: pipe
# Output: informaton read
# Read: Informaiton
`calcul/readpipe` := proc(pipeName, tries)
    local info, check;

    userinfo(5, 'MS', "Reading information from pipe", pipeName);
    if nargs = 2 then
        userinfo(6, 'MS', "Waiting", tries, "for pipe");
        if FAIL = block(tries, pipeName[1]) then
            userinfo(5,'MS',"Failed to read from pipe");
            RETURN();
        fi;
    else
        userinfo(6, 'MS', "Waiting forever for pipe", pipeName);
        if FAIL = block(5,pipeName[1]) then
            userinfo(5,'MS',"Failed to read from pipe", pipeName);
            RETURN();
        fi;
    fi;
    userinfo(6, 'MS', "Actually getting around to reading from pipe");
    info := readline(pipeName[1]);
    do
        check := traperror(parse(info));
        if check = lasterror then
            info := cat(info, readline(pipeName[1]));
        else
            break;
        fi;
    od;
    info := check;
    RETURN(info);
end:

# calcul/writepipe
#     Performs the writing of information to pipe
# Input: pipe, information
# Output: Error messages
# Write: Information
`calcul/writepipe` := proc(pipeName, info)
    local LineToWrite, Length, SubLine, LARGE, k, t;
    userinfo(5, 'MS', "Writing information to pipe", pipeName);
    LARGE := 10^8:
    LineToWrite := convert(info,string);
    Length := length(LineToWrite);
    for k from 1 to ceil(Length/LARGE) do
        SubLine := cat(LineToWrite[((k-1)*LARGE+1) ..
                min(Length,k*LARGE)], "\n");
        if FAIL = block(10,pipeName[2]) then
            print("Couldn't write to pipe");
            RETURN(-1);
        fi;
        t := fprintf(pipeName[2],SubLine);
    od;
    RETURN(t);
end:

# calcul/readpipe
#     Performs the reading of information from pipe
# Input: pipe
# Output: informaton read
# Read: Information
`calcul/readfile` := proc(fileName, tries)
    local info, check, fd, maxTries, good, i, ll;

    good := false;

    if nargs = 2 then maxTries := tries else maxTries := infinity fi;

    userinfo(5, 'MS', "Reading information from file", fileName);
    for i from 1 to maxTries do
        fd := traperror(open(fileName,READ));

        if fd = lasterror then
            traperror(close(fileName));
            next;
        fi;

        info := traperror(readline(fd));
        if info = lasterror then
            next;
        fi;

        check := traperror(parse(info));
        if check = lasterror then
            next;
        fi;

        ll := traperror(close(fd));

        do
            ll := system("rm -f ".fileName);
            if ll = -1 then
                print("It is not removing ".fileName." properly");
                print("Giving up");
                quit;
            fi;
            break;
        od;

        good := true;

        break;
    od;
    if good then
        info := check;
        RETURN(info);
    else
        RETURN(NULL);
    fi;
end:

# calcul/writefile
#     Performs the writing of information to file
# Input: file, information
# Output: Error messages
# Write: Information
`calcul/writefile` := proc(fileName, info, tries)
    local fd, t, maxTries, i;
    if nargs = 3 then
        maxTries := tries;
    else
        maxTries := infinity;
    fi;

    t := -1;
    userinfo(5, 'MS', "Writing information to file", fileName);
    for i from 1 to maxTries do
        fd := traperror(open(fileName,WRITE));
        if fd = lasterror then
            userinfo(5,'MS',fd);
            traperror(close(fileName));
#            if maxTries <> i then system("sleep 1"); fi;
            userinfo(6, 'MS', "Trying to write again");
            next;
        fi;

        t := writeline(fd, convert(info,string));
        traperror(close(fd));
        break;
    od;
    userinfo(6, 'MS', "Finished writing information to file", fileName);

    RETURN(t);
end:

# calcul/balancing/slave
#     The slave that does all the work
# Input: Recurrences
# Output:  -
# Read: What work to do, and other infomration
# Write: Information discovered, and what info is needed.
`calcul/balancing/slave` := proc(Known, readPipe, writePipe, Top, Bot, m, Q,
        slaveNumber)
    local Info, info, largest, j, i, s, Value, q;

    q := Q mod m;

    Info := copy(Known);

    userinfo(5,'MS',"Figuring out how much info is known", slaveNumber);
    for i from q to infinity by m do
        if has(Info[i], `Info`) then break; fi;
    od;
    largest := i - m;

    userinfo(5,'MS',"Knows info", seq(Info[m*i+q],i=0..(largest-q)/m));
    userinfo(5,'MS',"Largest known is", largest, slaveNumber);

    userinfo(5,'MS',"Figuring out s value");
    for i from 0 to infinity do
        if Bot(i) <> 0 then
            s := i;
            break;
        fi;
    od;

    do
        userinfo(3,'MS',"Slave ".slaveNumber." is waiting for instructions");
        do
            info := readpipe(readPipe);
            if info <> NULL then break; fi;
        od;

userinfo(5,'MS',"Got ", info, "from pipe");

# If has some info.  Now it has to figure out what it means

# If it is a calculation request.
if info[1] = "Work" then
    userinfo(1,'MS',"Slave ".slaveNumber." is working on determining".
        " the value for ". (info[2]));

    j := info[2];

    Value := Top(j+s):

    for i from q to largest by m do
        Value := Value - Bot(s+j-i)*Info[i]*binomial(j+s,i);
    od;

    userinfo(5,'MS',"Value, before asking master for help", Value);

    while largest+m < j do

        userinfo(3,'MS',"Asking for data of ", largest+m);
        writepipe(writePipe,["Need Data", largest+m]);

        do
            info := readpipe(readPipe);
            if info <> NULL then break; fi;
        od;

        if info[1] = "Data" then
            userinfo(3,'MS',"Got some data ".(info[2])." from "
                    .slaveNumber);

            userinfo(5,'MS',"Using this new data");
            Info[info[2]] := info[3];
            largest := info[2];
            Value := Value - Bot(s+j-largest)*Info[largest]*
                    binomial(j+s,largest);
            userinfo(5,'MS',"Value, after asking master for help",
                    Value);

        # Don't know what the hell it is doing.
        else
            print("What the hell is going on, waiting for data", info);
            quit;
        fi;

    od;

    Value := Value / binomial(j+s,s)/Bot(s);

    userinfo(2,'MS',"Telling the overseer about the new value for ". j);
    writepipe(writePipe,["Data", j, Value]);

elif info[1] = "Data" then
    userinfo(5,'MS',"Got new data", slaveNumber);

    Info[info[2]] := info[3];
    largest := info[2];

# Just quit
elif info[1] = "Quit" then
    userinfo(2, 'MS', "Slave Quitting", slaveNumber);
    close(readPipe[1]);
    close(readPipe[2]);
    close(writePipe[1]);
    close(writePipe[2]);
    RETURN();

# Don't know what the hell it is doing.
else
   print("What the hell is going on got", info, slaveNumber);
   quit;
```

```
        fi;
    od;
end:

# calcul/balancing/overseeerer
#     The communication on one machine
# Input: Recurrences
# Output:  -
# Read: What work to do, and other information
# Write: Information discovered, and what info is needed.
`calcul/balancing/overseer` := proc(Host, Me, Top, Bot, m, q, Known,
        numProcs, maxPipes)
    local readPipe, writePipe, info, numSlave, Info, slaveWait, slaveWork,
        Quit, slaveQuit, pid, i, j, workOn, messageSender, numProc, maxPipe, ll;

    workOn := []:
    if nargs >= 7 then
        Info := copy(Known);
    fi;
    if nargs = 9 then
        maxPipe := maxPipes;
    else
        maxPipe := 6;
    fi;
    if nargs >= 8 then
        numProc := numProcs;
    else
        numProc := 1;
    fi;

    numSlave := 0:

    writefile(cat(Me,2,Host), ["Need Work"]);

    do
        userinfo(3,'MS', "Waiting for instructions");
        info := NULL;
        do
            messageSender := 0:
            info := readfile(cat(Host,2,Me),1);
            if info <> NULL then break; fi;
            for messageSender from 1 to numSlave do
                info := readpipe(readPipe[messageSender],0);
                if info <> NULL then break; fi;
            od;
            if info <> NULL then break; fi;
        od;

        userinfo(1, 'MS', "Has ". numSlave. " slaves ".
                (numboccur([seq(slaveWork[i],i=1..numSlave)],true))
                ." running ". (numSlave - numboccur([seq(slaveWait[i],
                i=1..numSlave)],false)) ." waiting and the message is ".
                (info[1]));
        userinfo(5, 'MS', "Got info", info, "from ", messageSender);
        userinfo(3, 'MS', "Got info from slave/master ". messageSender);
        # Need to figure out what the message is

        # Find or create somebody to do the work
        if info[1] = "Work" then
            userinfo(1,'MS',"Told to do work on ".(info[2])." from ".
                    messageSender);

            if not has(Info[info[2]],'Info') then
                userinfo(2, 'MS', "Already know the info");
                writefile(cat(Me,2,Host), ["Data",info[2], Info[info[2]]]);
                Top(info[2]);
                Bot(info[2]);
                if workOn = [] then
                    writefile(cat(Me,2,Host), ["Need Work"]);
                fi;
                next;
            fi;

            for i from 1 to numSlave do
                if slaveWork[i] = false then break; fi;
            od;

            if i > maxPipe then
                workOn := [op(workOn),info[2]];

            # Create a new slave
            elif i > numSlave then
                userinfo(5,'MS',"Creating new slave",i,"to work on ",info[2]);
                numSlave := i;
                slaveWork[i] := true;
                slaveQuit[i] := false;
                slaveWait[i] := false;
                readPipe[i]  := pipe();
                writePipe[i] := pipe();
                Top(info[2]);
                Bot(info[2]);
                pid := fork();

                # The Slave
                if pid = 0 then
                    `calcul/balancing/slave`(Info, writePipe[i], readPipe[i],
                            Top, Bot, m, q, i);
                    quit;
                fi;

                writepipe(writePipe[i],info);

            # Use an old slave
            else
                userinfo(5,'MS',"Telling old slave ". i. " to work on ".
                        info[2]);
                slaveWork[i] := true;
                writepipe(writePipe[i],info);
            fi;

        # Check to see if the data is known
        # If it is, return it to the slave
        # If it isn't, put that slave in pending, and send off a need work
        elif info[1] = "Need Data" then

            userinfo(1,'MS', "Asked for data", info[2], "from", messageSender);

            # Doesn't know the information
            if has(Info[info[2]],Info) then
                userinfo(1,'MS',"Doesn't know the info", info[2], "for",
                        messageSender);
                slaveWait[messageSender] := info[2];

                if (numboccur([seq(slaveWork[l],l=1..numSlave)],true) -
                        (numSlave - numboccur([seq(slaveWait[l],
                        l=1..numSlave)], false))) < numProc and workOn = []
                        then
                    writefile(cat(Me,2,Host),["Need Work"]);
                    system("./sleepsm");
                fi;

            # It knows the information
            else
                userinfo(5,'MS',"Does know the info");
                writepipe(writePipe[messageSender],
                        ["Data",info[2],Info[info[2]]]);
            fi;

        # Deal with the data give overseer
        # Check to see if any slaves are waiting on it
        # If they are, make sure they get the information
        elif info[1] = "Data" then

            userinfo(1,'MS', "Given some new data ".(info[2])." from ".
                    messageSender);
            Info[info[2]] := info[3];
            for j from 1 to numSlave do
                if slaveWait[j] = info[2] then
```

```
                    userinfo(3, 'MS', "Telling waiting slave ". j. " about ".
                            "this data");

                    ll := writepipe(writePipe[j],
                            ["Data",info[2],Info[info[2]]]);
                    slaveWait[j] := false:
                fi;
            od;

            # If this data came from a slave, then we might need more
            # work for the slave to do, and tell the master.
            if messageSender <> 0 then
                slaveWork[messageSender] := false;
                writefile(cat(Me,2,Host),["Data",info[2],info[3]]);
                if workOn = [] then
                    userinfo(2,'MS',"Slave ". messageSender.
                            " is no longer working");
                    if (numboccur([seq(slaveWork[l],l=1..numSlave)], true) -
                            (numSlave - numboccur([seq(slaveWait[l],
                            l=1..numSlave)], false))) <
                            numProc then
                        userinfo(2,'MS',"Ask for more work");
                        writefile(cat(Me,2,Host),["Need Work"]);
                    fi;
                else
                    userinfo(2,'MS',"Slave", messageSender,
                            "is no longer working, ",
                            "so give it outstanding work");
                    writepipe(writePipe[messageSender],
                            ["Work",workOn[1]]);
                    workOn := workOn[2..-1];
                    slaveWork[messageSender] := true;
                fi;
            fi;

        # Doesn't want to give any more work.
        elif info[1] = "Quit" then
            for i from 1 to numSlave do
                if slaveWork[i] = false and slaveQuit[i] = false then
                    userinfo(2,'MS',"Telling the ".i."th slaves to quit");
                    slaveQuit[i] := true;
                    writepipe(writePipe[i],["Quit"]);
                fi;
            od;
            for i from 1 to numSlave do
                if slaveQuit[i] = false then break; fi;
                userinfo(2,'MS',"The ".i."th slave has quit");
            od;
            if i > numSlave then
                userinfo(1,'MS',"Everyones quit, time to go home");
                for i from 1 to numSlave do
                    close(writePipe[i][1]);
                    close(writePipe[i][2]);
                    close(readPipe[i][1]);
                    close(readPipe[i][2]);
                od;
                RETURN();
            fi;

        # Don't know what the hell happened
        else
            RETURN("What the hell just happened");
            quit;
        fi;
    od;
end:

# calcul/balancing/master
#     The main controller of all things good.
# Input: Nothing of importance
# Output:  -
# Read: Just about everything (the master knows all)
# Write: Just about anything (the master can order around all)
# Reference: Section 6.1.
`calcul/balancing/master` := proc(Host, Mach, Largest, m, q, fileName,
    interval, Known)

    local Info, info, i, j, k, maxKnown, needToWrite, writeThis, mach, l, fn,
            pid;

    Info := copy(Known);
    maxKnown := -1;

    mach := Mach;

    for i in Mach do
        needToWrite[i] := []:
    od;

    i := q;

    while Largest > maxKnown do
        info := NULL;
        userinfo(3,'MS',"Waiting for instructions");
        do
            for l from 1 to nops(mach) do
                j := mach[l];
                info := readfile(cat(j,2,Host), 1);

                userinfo(4,'MS',"Checking to see if there are outstanding ".
                        "messages for", j);
                if needToWrite[j] <> [] then
                    writeThis := needToWrite[j][1];
                    userinfo(5,'MS',"Sending information ".
                            "again to", j);
                    if (writefile(cat(Host,2,j),
                            writeThis, 1) <> -1) then
                        needToWrite[j] := needToWrite[j][2..-1];
                    fi;
                fi;
                if info <> NULL then
                    mach := [seq(mach[k],k=l+1..nops(mach)),
                            seq(mach[k],k=1..l)];
                    break;
                fi;
            od;
            if info <> NULL then break; fi;
            system("./sleepsm");
        od;

        userinfo(5,'MS', "Got information", info, "from", j);
        # We have info from one of the over seers, we have to
        # now figure out what it is.

        # Check to see if it is a request for work
        if info[1] = "Need Work" then
            userinfo(1,'MS', "Working on requested for work from ". j);

            if i > Largest then
                userinfo(2,'MS', "Tell ".j." to quit");
                if writefile(cat(Host,2,j),["Quit"],1) = -1 then
                    needToWrite[j] := [op(needToWrite[j]),["Quit"]]:
                fi;
            else
                userinfo(2,'MS', "Tell ".j." to work on the value of ". i);
                # Here I HAVE to make sure that they have had all
                # previous messages first.
                if needToWrite[j] = [] then
                    if writefile(cat(Host,2,j),["Work",i],1) = -1 then
                        needToWrite[j] := [op(needToWrite[j]),["Work",i]]:
#                        system("sleep 1");
                    fi;
                else
                    needToWrite[j] := [op(needToWrite[j]),["Work",i]]:
                fi;
            fi;
            i := i + m;
        # Check to see if it is new info
        elif info[1] = "Data" then
            userinfo(1,'MS', "Got some data for the value of ".(info[2]).
                    " from ". j);
            Info[info[2]] := info[3];

            maxKnown := max(maxKnown, info[2]);

            for k in Mach do
                if j = k then next; fi;
                userinfo(3,'MS', "Telling ". k. " about information");
                if writefile(cat(Host,2,k),
                        ["Data",info[2],info[3]],1) = -1 then
                    needToWrite[k] := [op(needToWrite[k]),
                            ["Data",info[2],info[3]]];
#                    system("sleep 1");
                fi;
            od;

            if (info[2] = 0) mod interval then
                fn := fileName.(info[2]).`.m`;
                pid := fork();
                if pid = 0 then
                    save Info, fn;
                    quit;
                fi
            fi;

        # Don't know what it is, make an error
        else
            print("What the hell is going on II got", info);
            quit;
        fi;
    od;
    for k in Mach do
        userinfo(1,'MS', "Telling ". k. " to quit");
        writefile(cat(Host,2,k),["Quit"],1);
    od;

    RETURN(op(Info));
    # Need to tell people to quit still.
end:

#libname := libname[3], libname[1..2]:
savelib(`calcul/readpipe`,  `calcul/readpipe.m`);
savelib(`calcul/writepipe`, `calcul/writepipe.m`);
savelib(`calcul/readfile`, `calcul/readfile.m`);
savelib(`calcul/writefile`, `calcul/writefile.m`);
savelib(`calcul/balancing/slave`,  `calcul/balancing/slave.m`);
savelib(`calcul/balancing/overseer`, `calcul/balancing/overseer.m`);
savelib(`calcul/balancing/master`,  `calcul/balancing/master.m`);
```

Bibliography

[1] *Cecm research projects*, http://www.cecm.sfu.ca/projects, 1999.

[2] Milton Abramowitz and Irene A. Stegun (eds.), *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, Dover Publications Inc., New York, 1992, Reprint of the 1972 edition.

[3] J. L. Adams, *Conceptual blockbusting: A guide to better ideas*, Freeman, San Francisco, 1974.

[4] Bruce C. Berndt, *Ramanujan's notebooks*, Springer-Verlag, New York, 1994.

[5] Jonathan Borwein, Peter Borwein, and Lennart Berggren, *Pi: A source book*, Springer, New York, 1997.

[6] Jonathan M. Borwein, David M. Bradley, and Richard E. Crandall, *Computational strategies for the Riemann zeta function*, (unpublished), 1996.

[7] Carl B. Boyer, *A history of mathematics*, John Wiley & Sons, Inc., 1968.

[8] L Carlitz, *Some arithmetic properties of the oliver functions.*, Mathematische Annalen **128** (1955), 412 – 419.

[9] Mustapha Chellali, *Accélération de calcul de nombres de Bernoulli*, Journal of Number Theory (1988), 347–362.

[10] Louis Comtet, *Advanced combinatorics, the art of finite and infinite expansions*, D. Reidel Publishing Company, Boston, 1974.

[11] F. N. David, M. G. Kendall, and D. E. Barton, *Symmetric function and allied tables*, Cambridge, Cambridge, 1966.

[12] K.O. Geddes, S.R. Czapor, and G. Labahn, *Algorithms for computer algebra*, Kluwer Academic Publishers, 1996.

[13] K.O. Geddes, G. Labahn, M. B. Monagan, and S. Vorketter, *The Maple programming guide*, Springer-Verlag, New York, 1996.

[14] J. W. L. Glaisher, *On Eulerian numbers*, Quarterly Journal of Mathematics **45** (1914).

[15] Gene H. Golub and Charles F. van Loan, *Matrix computations*, second ed., The Johns Hopkins University Press, Baltimore, 1989.

[16] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete mathematics*, second ed., Addison-Wesley Publishing Company, Reading, MA, 1994, A foundation for computer science.

[17] G. H. Hardy and W. M Wright, *An introduction to the theory of numbers*, fourth ed., Clarendon Press, Oxford, 1960.

[18] I. N. Herstein, *Topics in algebra*, second ed., Xerox College Publishing, Lexington, Mass., 1975.

[19] D.H. Lehmer, *Lacunary recurrence formulas for the numbers of Bernoulli and Euler*, Annals of Mathematics **36** (1935), no. 3, 637–649.

[20] Maurice Mignotte, *Mathematics for computer algebra*, Springer-Verlag, New York, 1992, Translated from the French by Catherine Mignotte.

[21] J. Miller, N. J. A. Sloane, and N. E. Young, *A new operation on sequences: the boustrophedon transform*, J. Combn Theory **17A** (1996), 44–54.

[22] S Ramanujan, *Some properties of Bernoulli's numbers*, Indian Mathematical Journal (1911).

[23] J. Riordan, *An introduction to combinatorial analysis*, Wiley, 1958.

[24] John Riordan, *Combinatorial identities*, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons, New York, 1968.

[25] N. J. A. Sloane and Simon Plouffe, *The encyclopedia of integer sequences*, Academic Press, Toronto, 1995.

[26] Neil J. A. Sloane, *Sloane's on-line encyclopedia of integer sequences*, http://akpublic.research.att.com/∼njas/sequences/index.html, 1998.

[27] C. R. Snow, *Concurrent programming*, Cambridge Computer Science Texts, no. 26, Cambridge University Press, New York, 1992.

[28] I Steward, *Math. rec.*, Scientific American (1996).

[29] W. A. Whitworth, *Dcc exercises in choice and chance*, Stechert, New York, 1945.

[30] Herbert S Wilf, *Generating functionology*, Academic Press, Inc., Toronto, 1990.