GOSPER'S ALGORITHM -
A DECISION PROCEDURE FOR PARTIAL
HYPERGEOMETRIC SUMS

by

Richard B. Reeves

B.A., University of Colorado at Denver, 1990

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado at Denver
in partial fulfillment
of the requirements for the degree of
Master of Science
Applied Mathematics

1992
This thesis for the Master of Science degree by

Richard B. Reeves

has been approved for the

Department of

Mathematics

by

William Cherowitzo

Stanley E. Payne

Sylvia Lu

Date
Reeves, Richard B. (M.S., Applied Mathematics)

Gosper's Algorithm

A Decision Procedure for Hypergeometric Sums

Thesis directed by Associate Professor William Cherowitzo

Given an integer function $f_k$, does there exist a function $S$ such that

$$
\sum_{k=1}^{m} f_k = S(m) - S(0) \quad \text{or, equivalently} \quad f_k = S(k) - S(k-1) \quad ?
$$

An algorithm created by R.W. Gosper in 1978 gives a complete answer to this question. In creating this algorithm Gosper has made a decisive step in the mechanization of hypergeometric identity proofs.

The form and content of this abstract are approved. I recommend its publication.

Signed __________________________
William Cherowitzo
To my parents Jim and Millie

for their unwavering love, support and encouragement

and

To Terry

for her friendship, intellect, humor, and for

her love - that compels me to see the beauty in life.
ACKNOWLEDGEMENTS

I would like to thank Bill Cherowitzo, Stan Payne, and Zenas Hartvigson for sharing their expertise. Above all, I would like to thank Dr. Cherowitzo for his guidance, support, and patience.
# TABLE OF CONTENTS

Introduction ...................................................... 1

Chapter 1: Preliminaries ........................................ 2
  1.1 Hypergeometric Functions .......................... 2
  1.2 Hypergeometric Terms ................................. 6
  1.3 General Method for Finding Hypergeometric Representation ........ 8

Chapter 2: Gosper's Algorithm ............................... 14
  2.1 Partial Hypergeometric Sums ..................... 14
  2.2 Algorithmic Method ................................. 15

Chapter 3: Mechanized Proofs ............................... 24

Appendix

Hypergeometric Program ................................. 29

References ....................................................... 55
Introduction

Let \( f_k \) be a complex values function of the integer \( k \). Given such a function \( f_k \), does there exist a function \( S \) (within an additive constant) such that

\[
\sum_{k=1}^{n} f_k = S(m) - S(0)
\]

or, equivalently

\[
f_k = S(k) - S(k-1)
\]

Such a sum of the function \( f \) in finite calculus would be called an "indefinite sum", in analogy with the indefinite integral of infinite calculus. An algorithm created by R. W. Gosper in 1978 gives a complete answer to this question. In other words Gosper’s algorithm will either return the function \( S \) (with the desired properties), or it will guarantee that no such function exists. It never returns the statements "not sure" or "unknown" to this question. This paper examines Gosper’s Algorithm.
Chapter 1: Preliminaries

1.1 Hypergeometric Functions

Gosper has discovered a way to decide whether a given function is partially (indefatibly) summable with respect to a general class of functions called hypergeometric terms. Therefore, in order to investigate Gosper’s algorithm, it is first necessary to understand hypergeometric series or functions.

Hypergeometric (HG) series were first investigated by Euler, Gauss, Pfaff, and Riemann. The study of HG series has evolved into a systematic means of handling a great variety of binomial coefficient identities [5].

The following notation will be useful in the discussion of HG series.

Definition 1.1: Rising Factorial Powers

\[ x^{ar{m}} = x(x+1)...(x+m-1) \quad \text{integer} \quad m \geq 0. \]

Definition 1.2: Falling Factorial Powers

\[ x^\underline{m} = x(x-1)...(x-m+1) \quad \text{integer} \quad m \geq 0. \]
The general hypergeometric series is a power series in \( z (z \in \mathbb{C}) \), and is defined as follows in terms of rising factorial powers:

**Definition 1.3:**

\[
F \left( \begin{array}{c}
a_1, \ldots, a_m \\
b_1, \ldots, b_n
\end{array} \mid z \right) = \sum_{k=0}^{\infty} \frac{a_1^k \cdots a_m^k}{b_1^k \cdots b_n^k} \frac{z^k}{k!}
\]

where, to avoid division by zero, none of the \( b \) parameters may be zero or a negative integer. Using standard conventions, we define \( x^0 = 1 \) for all \( x \) and \( 0^0 = 1 \). Therefore, the first term \((k = 0)\) of any HG series is equal to 1.

Many important functions occur as some special case of the general hypergeometric series. This is one of the reasons hypergeometric functions are so powerful. The following examples illustrate a small portion of this power.

**Example 1.1** (The simplest situation - no parameters):

\[
m = 0, \quad n = 0
\]

\[
F \left( \begin{array}{c}

\end{array} \mid z \right) = \sum_{k=0}^{\infty} \frac{z^k}{k!} = e^z.
\]

This "blank" notation does make sense, when \( m \) or \( n \) equals zero, because the notation indicates the absence of parameters. However, this form has proven to be perplexing to many. In order to avoid confusion, it has
become standard to add an extra one above and below whenever \( m = 0 \) or whenever \( n = 0 \). So,

\[
F \left( \frac{1}{1} \right) = \sum_{k=0}^{\infty} \frac{z^k}{k!} = \sum_{k=0}^{\infty} \frac{1}{k!} \frac{z^k}{k!} = F \left( \frac{1}{1} \right).
\]

It is now important to note that, when using HG series in Gosper's algorithm, we are not concerned with convergence of the series. We are simply using \( z \) as a formal symbol. Therefore, any identities derived will be formally true. An example of this is the HG function \( F \left( \frac{1}{1} \right) = k! z^k \).

This series doesn't converge for any nonzero \( z \) but it is still very useful in solving problems [5].

**Example 1.2** (specific example):

\[
m = 1, \quad n = 0, \quad a_1 = 1 \quad \text{written as} \quad m = 2, \quad a_1 = a_2 = 1, \quad \text{and} \quad n = 1, \quad b_1 = 1
\]

\[
F \left( \frac{1}{1} \right) = F \left( \frac{1}{1} \right) = \sum_{k=0}^{\infty} z^k = \frac{1}{1-z}.
\]

This is the geometric series. Hypergeometric series are so named because the geometric series is included as a very specific case.
Example 1.3 (general example):

\[
m = 1, \ n = 0, \\
F\left(\frac{a,1}{1} \mid z\right) = \sum_{k=0}^{\infty} \frac{a^k}{k!} z^k = \sum_{k=0}^{\infty} \frac{(a)(a+1)\cdots(a+k-1)}{k!} z^k \\
= \sum_{k=0}^{\infty} \left(\frac{a+k-1}{k}\right) z^k = \frac{1}{(1-z)^a}.
\]

Example 1.4:

\[
F\left(-\frac{a,1}{1} \mid -z\right) = \sum_{k=0}^{\infty} \frac{(-a)^k}{k!} (-z)^k = \sum_{k=0}^{\infty} \frac{(a)(a-1)\cdots(a-k+1)}{k!} z^k \\
= \sum_{k=0}^{\infty} \left(\frac{a}{k}\right) z^k = (1+z)^a.
\]

This utilizes the binomial theorem. Note, a negative integer as an upper parameter causes the infinite series to become finite - since \((-a)^k = 0\) whenever \(k > a \geq 0\) and \(a\) is an integer.

Example 1.5 (The Gauss Hypergeometric - so named because many of its properties were first proved by Gauss in his doctoral dissertation in 1812; although Euler and Pfaff had already discovered many of these properties [5]):

\[
F\left(\frac{a,b}{c} \mid z\right) = \sum_{k=0}^{\infty} \frac{a^k b^k}{c^k k!} z^k.
\]
Before everything was generalized to arbitrary \( m \) and \( n \) (around 1870), this was considered the hypergeometric series.

One important special case of the Gauss Hypergeometric is:

\[
\ln(1 + z) = z \begin{pmatrix} 1,1 \\ 2 \end{pmatrix} = \sum_{k \geq 0} \frac{k!k!}{(k+1)!} \frac{(-z)^k}{k!} = \sum_{k \geq 0} \frac{(-z)^k}{(k+1)} = z \left( 1 + \frac{-z}{2} + \frac{z^2}{3} + \ldots \right) = z - \frac{z^2}{2} + \frac{z^3}{3} - \frac{z^4}{4} + \ldots.
\]

Note: \( z^1 \ln (1 + z) \) is a hypergeometric function but \( \ln (1 + z) \) is not, since by convention every hypergeometric function has a value of 1, when \( z = 0 \).

1.2 Hypergeometric Terms

It is essential to realize which series can be expressed as a hypergeometric series. This determination is made by examining the ratio of consecutive terms of any hypergeometric series. Let

\[
F \left( \begin{array}{c} a_1, \ldots, a_m \\ b_1, \ldots, b_n \end{array} \mid z \right) = \sum_{k \geq 0} t_k \quad \text{where} \quad t_k = \frac{a_{1}^{k} \ldots a_{m}^{k}}{b_{1}^{k} \ldots b_{n}^{k}} \frac{z^{k}}{k!}.
\]

As previously stated \( t_0 = 1 \), and all other terms have the following ratio:
\[
\frac{t_{k+1}}{t_k} = \frac{a_1 \cdots a_{m}}{a_1 \cdots a_m} \frac{b_1 \cdots b_n}{b_1 \cdots b_n} \frac{z^{k+1}}{(k+1)!} \frac{k!}{z^k}
\]

\[
= \frac{(a_1 + k) \cdots (a_m + k)}{(b_1 + k) \cdots (b_n + k)} \frac{z}{k+1}.
\]

Therefore, a hypergeometric series is a series \( \sum_{k=0}^\infty t_k \) in which the ratio of every two consecutive terms is a rational function of the summation variable \( k \). This form exhibits the fact that the \( a \)'s are the negatives of the roots of the numerator and the \( b \)'s are negatives of the roots of the denominator of the term ratio. In order to maintain a canonical form for the term ratio, if \( (k+1) \) is not a factor in the denominator, we add it to both the numerator and the denominator.

To restate, hypergeometric series are those series whose constant term (first term) is 1 and whose term ratio is a rational function of the summand variable. Given a rational function, we may always find a hypergeometric series whose term ratio is this function.

**Example 1.6:**

\[
\frac{t_{k+1}}{t_k} = \frac{k^2 - 7k + 12}{8k^2 + 1} = \frac{(k-3)(k-4)(1/8)}{(k+i/4)(k-i/4)}
\]

adding the missing \( (k+1) \) we get
\[
\frac{(k-3)(k-4)(1/8)(k+1)}{(k+i/4)(k-i/4)(k+1)}.
\]

This is the term ratio of

\[ F\left( -3, -4, 1 \mid \frac{1}{8} \right). \]

1.3 General Method for Finding Hypergeometric Representation of a Given Function \( S \).

Step 1) Write \( S = \sum_{k=0} t_k \cdot t_0 \neq 0 \).

Step 2) Calculate the term ratio.

Step 3) If term ratio is not a rational function of \( k \), the series cannot be expressed in hypergeometric form.

Step 4) Otherwise, find the negatives of the roots of the numerator and denominator and express in the following form:

\[
\frac{(a_1+k) \cdots (a_m+k)}{(b_1+k) \cdots (b_n+k)} \cdot \frac{z}{k+1}.
\]

This gives the parameters \( a_1, \ldots, a_m; b_1, \ldots, b_n; \) and the argument \( z \) such that
$$S = t_0 \ F\left(\frac{a_1, \ldots, a_m}{b_1, \ldots, b_n} | z \right).$$

**Example 1.7** \( (S = \sin(z)):\)

**Step 1)**

\[ \sin(z) = z - \frac{z^3}{3!} + \frac{z^5}{5!} - \frac{z^7}{7!} + ... \]

\[ \sin(z) = \sum_{k=0}^{\infty} \frac{(-1)^k z^{2k+1}}{(2k+1)!}, \quad \text{first term is } z. \]

\[ z^{-1} \sin(z) = \sum_{k=0}^{\infty} \frac{(-1)^k z^{2k}}{(2k+1)!}, \quad \text{first term is } 1. \]

**Step 2)**

\[ \frac{t_{k+1}}{t_k} = \frac{(-1)^{k+1} z^{2k+2}}{(-1)^k z^{2k}} \frac{(2k+1)!}{(2k+3)(2k+2)} = \frac{(-1)^k z^2}{(2k+3)(2k+2)}. \]

**Step 4)**

\[ \frac{t_{k+1}}{t_k} = \frac{-z^2}{4 \ (k+3/2)(k+1)} \]

\[ \sin(z) = z \ F\left(\frac{1}{1,3/2} | \frac{-z^2}{4} \right). \]
One of the strengths of hypergeometric series is the ability to express and manipulate a great variety of binomial summations in a systematic way.

Example 1.8:

A well known binomial coefficient identity is the parallel summation law:

\[
\sum_{k=0}^{n} \binom{r+k}{k} = \binom{r}{0} + \binom{r+1}{1} + \ldots + \binom{r+n}{n} = \binom{r+n+1}{n}, \quad \text{integer } n.
\]

If we wished to express the parallel summation law in its hypergeometric form we would proceed as follows:

Step 1)

\[
\sum_{k=0}^{n} \binom{r+k}{k} = \binom{r+n+1}{n}, \quad \text{integer } n.
\]

If we replace \( k \) with \( n-k \) we get

\[
\sum_{n-k=0}^{n} - \sum_{k=0}^{n} \binom{r+n-k}{n-k} = \sum_{k=0}^{n} \frac{(r+n-k)!}{(n-k)! \ r!},
\]

where the first term \( \binom{r+n}{n} = \frac{(r+n)!}{n! \ r!} \).
Step 2)

\[
\frac{t_{k+1}}{t_k} = \frac{(r+n-k-1)! \, (n-k)! \, r!}{(r+n-k)! \, (n-k-1)! \, r!} = \frac{(n-k)}{(r+n-k)}. \]

Step 4)

\[
\frac{(n-k)}{(r+n-k)} = \frac{(n-k) \, (k+1)}{(r+n-k) \, (k+1)}
\]

\[
\binom{r+n}{n} \, F \left( \begin{array}{c}
-n,1 \\
-(r+n)
\end{array} \mid 1 \right) = \binom{r+n+1}{n}.
\]

Solving for the hypergeometric function in this equation, we obtain

\[
F \left( \begin{array}{c}
-n,1 \\
-(r+n)
\end{array} \mid 1 \right) = \frac{r+n+1}{r+1}, \quad \text{if} \quad \binom{r+n}{n} \neq 0.
\]

Note that since the lower parameters of a hypergeometric cannot be zero or negative integers, this represents a degenerate case of the parallel summation law because parallel summation is usually applied when \( r \) and \( n \) are positive integers. This limitation is bypassed in this and other situations where it arises by considering:

\[
\lim_{\varepsilon \to 0} F \left( \begin{array}{c}
1,-n \\
-n-r+\varepsilon
\end{array} \mid 1 \right).
\]
Example 1.9 (Vandermonde’s Convolution Identity):

\[
\sum_{k \geq 0} \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n}.
\]

A combinatorial description of this identity is the sum of all possible ways to pick \( k \) men from a total of \( r \) men and the number of ways to pick \( n-k \) women from a total of \( s \) women equals the number of ways to choose \( n \) people from a group consisting of \( r \) men and \( s \) women.

Let

\[
t_k = \binom{r}{k} \binom{s}{n-k} = \frac{r! \cdot s!}{k!(r-k)!(n-k)!(s-n+k)!}.
\]

We can make several general observations about the process of converting to hypergeometric form [5].

1) Whenever \( t_k \) contains a factor like \((\alpha+k)!\), with a plus sign before the \( k \), we get \( \frac{(\alpha+k+1)!}{(\alpha+k)!} = (\alpha+k+1) \) in the term ratio. Which contributes parameter \((\alpha+1)\) to the corresponding HG function, as an upper parameter if \((\alpha+k)!\) was in the numerator of \( t_k \), or as a lower parameter if it was in the denominator of \( t_k \).

2) Whenever \( t_k \) contains a factor like \((\alpha-k)!\), with a minus sign before the \( k \), we get \( \frac{(\alpha-k-1)!}{(\alpha-k)!} = \frac{(-1)}{(k-\alpha)} \) in the term ratio. Which contributes
Chapter 2: Gosper’s Algorithm

2.1 Partial Hypergeometric Sums

We have seen how hypergeometric sums can be used to systematically express a large number of binomial coefficient sums. It would appear that partial sums may be needed in many instances (combinatorial solutions, physical problems, and computational complexity determination to name a few). When dealing with such sums it is often desirable to find a closed form solution that works over a general range. Gosper’s Algorithm decides whether a given summand (function) is partially summable (within an additive constant) with respect to a general class of functions called hypergeometric terms by examining

$$\sum_{k=1}^{n} f_k = S(m) - S(0)$$

or, equivalently

$$f_k = S(k) - S(k-1). \quad (1)$$

Given $f_k$ the algorithm finds those $S(k)$ with the property

$$\frac{S(k)}{S(k-1)} = a \text{ rational function of } k .$$

2.2 Algorithmic Method [4]
Step 1

If \( \frac{S(k)}{S(k-1)} \) is a rational function of \( k \) then the term ratio

\[
\frac{f_k}{f_{k-1}} = \frac{S(k) - S(k-1)}{S(k-1) - S(k-2)} = \frac{S(k)}{S(k-1)} - 1
\]

is also a rational function of \( k \). Therefore, we can express the term ratio of (1) as

\[
\frac{f_k}{f_{k-1}} = \frac{p_k}{p_{k-1}} \cdot \frac{q_k}{r_k}
\]  

(2)

where \( p_k, q_k, \) and \( r_k \) are polynomials in \( k \) subject the following condition:

\[
(k+\alpha) \mid q_k \quad \text{and} \quad (k+\beta) \mid r_k
\]

\[
\Rightarrow \alpha - \beta \text{ is not a nonnegative integer.}
\]  

(3)

It is always possible to achieve this condition with a rational function with the following steps:

a) Since the term ratio is a rational function of \( k \), we can express this ratio in hypergeometric form,

\[
\frac{f_k}{f_{k-1}} = \frac{(a_1+k) \cdots (a_m+k)}{(b_1+k) \cdots (b_n+k)} z
\]

b) Set:
\[ \begin{align*}
    p_{k-1} &= 1, \\
    q_k &= (k+a_1) \cdots (k+a_m) z, \quad \text{and} \\
    r_k &= (k+b_1) \cdots (k+b_n).
\end{align*} \]

c) If (3) is violated then go to d).

d) If \( q \) and \( r \) have factors \((k + \alpha)\) and \((k + \beta)\) respectively that violate (3), then divide them out of \( q \) and \( r \) and replace \( p_k \) by

\[ p_k(k+\alpha)^N = p_k(k+\alpha)(k+\alpha-1) \cdots (k+\beta+1) \quad \text{where} \quad N = \alpha - \beta. \]

The new \( p, q, \) and \( r \) still satisfy (2) since \( \frac{p_k}{p_{k-1}} \) is replaced by

\[ \frac{p_k}{p_{k-1}} \frac{(k+\alpha)}{(k+\beta)}. \]

e) Repeat step c) until condition (3) is satisfied.

**Step 2**

Recall the goal is to find some hypergeometric term \( S(k) \) such that (1) is satisfied. Assume

\[ f_k = S(k) - S(k-1) \]

and now write

\[ S(k) = \frac{q_{k+1}}{p_k} g(k) f_k \quad \text{(4)} \]
where \( g(k) \) is a function to be found.

Combining (1) and (4) we get

\[
g(k) = \frac{p_k}{q_{k+1}} \frac{S(k)}{S(k) - S(k-1)} = \frac{p_k}{q_{k+1}} \frac{1}{1 - \frac{S(k-1)}{S(k)}}.
\]

This implies \( g(k) \) is a rational function of \( k \) whenever \( S(k) / S(k-1) \) is.

By substituting (4) into (1) we get

\[
f_k = \frac{q_{k+1}}{p_k} g(k) \frac{f_k}{f_{k-1}} - \frac{q_k}{p_{k-1}} g(k-1) \frac{f_{k-1}}{f_k}.
\]

Multiplying through by \( p_k/f_k \) and using (2) (to solve for \( r_k \)) we get

\[
p_k = q_{k+1} g(k) - r_k g(k-1).
\]

If we can find a \( g(k) \) that satisfies this recurrence, we've found \( S(k) \) and we are done.

**Theorem:** If \( S(k) / S(k-1) \) is a rational function of \( k \), then \( g(k) \) is a polynomial.

**Proof:** We know that \( g(k) \) is a rational function when \( S(k) / S(k-1) \) is, so suppose by way of contradiction

\[
g(k) = \frac{u(k)}{v(k)} \quad \text{where} \quad v(k) \neq 1
\]
and \( u(k) \) and \( v(k) \) have no factors in common. Let \( N \) be the largest integer such that \((k+\beta) \) and \((k+\beta+N) \) both occur as factors of \( v(k) \) for some \( \beta \in \mathbb{C} \). The value of \( N \) is nonnegative, since \( N = 0 \) will always satisfies this condition. Substituting into equation (5) we get

\[
p_k = q_{k+1} \frac{u(k)}{v(k)} - r_k \frac{u(k-1)}{v(k-1)}.
\]

Now, equation (5) can be rewritten

\[
p_k v(k) v(k-1) = q_{k+1} u(k) v(k-1) - r_k u(k-1) v(k).
\]

If we set \( k = -\beta + 1 \) and \( k = -\beta - N \) we get

\[
r(-\beta + 1) u(-\beta) v(-\beta + 1) = 0
\]

and

\[
q(-\beta - N + 1) u(-\beta - N) v(-\beta - N - 1) = 0.
\]

Now \( u(-\beta) \neq 0 \) and \( u(-\beta - N) \neq 0 \), because \( u \) and \( v \) have no factors in common. Also, \( v(-\beta + 1) \neq 0 \) and \( v(-\beta - N - 1) \neq 0 \), because \( v(k) \) would otherwise contain the factor \((k+\beta-1) \) or \((k+\beta+N+1) \) which is contrary to \( N \) being maximal. Therefore, \( r(-\beta + 1) = 0 = q(-\beta - N + 1) \). But this contradicts condition (3). Hence \( g(k) \) must be a polynomial [5].
Step 3

All that remains is finding, given $p_k$, $q_k$, and $r_k$, if there exists a polynomial $g(k)$ that satisfies equation (5). This turns out to be surprisingly easy, once we know the degree $d$ of $g(k)$.

Let

$$g(k) = a_d k^d + a_{d-1} k^{d-1} + \ldots + a_0, \quad a_d \neq 0$$  \hspace{1cm} (6)

Rewrite equation (5) in the following form:

$$2p(k) = \text{Diff}Q(k) \times (g(k) + g(k-1)) - \text{Sum}Q(k) \times (g(k) - g(k-1))$$

where

$$\text{Diff}Q(k) = q(k+1) - r(k) \quad \text{and} \quad \text{Sum}Q(k) = q(k+1) + r(k).$$ \hspace{1cm} (7)

If $g(k)$ has degree $d$ then sum $(g(k) + g(k-1)) = 2a_d k^d + \ldots$ also has degree $d$. Whereas the difference $(g(k) - g(k-1)) = d a_d k^{d-1} + \ldots$ has degree $d-1$ (assume the zero polynomial has degree -1.) Write deg(p) for the degree of any polynomial $p(k)$.

Case 1: The $\text{deg}(\text{Diff}Q) \geq \text{deg}(\text{Sum}Q)$. This implies the degree of the right-hand side of (7) is $\text{deg}(\text{Diff}Q) + d$, so $d = \text{deg}(p) - \text{deg}(\text{Diff}Q)$.

Case 2: The $\text{deg}(\text{Diff}Q) < \text{deg}(\text{Sum}Q)$. This implies the we can write
\[ \text{Diff} Q(k) = \beta k^{d'-1} + \ldots \]

and

\[ \text{Sum} Q(k) = \gamma k^{d'} + \ldots \quad \text{where} \quad \gamma \neq 0. \]

The right-hand side of (7) now has the form
\[(2\beta a_d + \gamma d a_d)k^{d-d'-1} + \ldots \]. This leaves two possibilities:

1) \[2\beta + \gamma d \neq 0 \quad \Rightarrow \quad d = \text{deg}(p) - \text{deg}(\text{Sum} Q) + 1,\]

or

2) \[2\beta + \gamma d = 0 \quad \Rightarrow \quad d > \text{deg}(p) - \text{deg}(\text{Sum} Q) + 1.\]

This second case needs to be examined only if \(-2\beta / \gamma\) is an integer \(d\) greater than \(\text{deg}(p) - \text{deg}(\text{Sum} Q) + 1\).

If \(d < 0\), the algorithm stops with the message that no function \(S\), in equation (1), exists. If a nonnegative \(d\) is discovered, we can plug equation (6) into equation (5). The polynomial \(g(x)\) will satisfy the recurrence if and only if the \(\alpha\)'s of equation (6) satisfy certain linear equations, because each power of \(k\) must have the same coefficient on both sides of (5).

**Example 2.1** (This example was computed using the **Derive** software package on an IBM compatible 486 PC):
Use Gosper's algorithm to find a closed form identity for the following summation:

\[
\sum_{k=1}^{m} \frac{\prod_{j=1}^{k-1} aj^2 + bj + c}{\prod_{j=1}^{k+1} aj^2 + bj + d}
\]

Step 1

\[
f(k) = \frac{\prod_{j=1}^{k-1} aj^2 + bj + c}{\prod_{j=1}^{k+1} aj^2 + bj + d}
\]

\[
\frac{f(k)}{f(k-1)} = \frac{\prod_{j=1}^{k-1} aj^2 + bj + c}{\prod_{j=1}^{k+1} aj^2 + bj + d} \cdot \frac{\prod_{j=1}^{k} aj^2 + bj + d}{\prod_{j=1}^{k-2} aj^2 + bj + c}
\]

\[
= \frac{a(k-1)^2 + b(k-1) + c}{a(k+1)^2 + b(k+1) + d}
\]

Step 2

Let

\[
q(k) = a(k-1)^2 + b(k-1) + c
\]

\[
r(k) = a(k+1)^2 + b(k+1) + d
\]
and suppose \( q \) and \( r \) have no factors in common. This implies \( p_k = 1 \).

Therefore,

\[
\text{deg}(q_{k+1} - r_k) = 1 < 2 = \text{deg}(q_{k+1} + r_k)
\]

which requires Case 2, where we discover

\[
-2p/\gamma = 2 > 1 = \text{deg}(p) - \text{deg}(\text{SumQ}) + 1.
\]

So \( g \) will be a quadratic function of \( k \) with three undetermined coefficients.

Let \( g(k) = g_2k^2 + g_1k + g_0 \)

Step 3

Substituting into equation (5)

\[
1 = k^2 \left( g_2(2a+b+c-d) + g_1(-a) \right) + k \left( g_2(b+2d) + g_1(a+c-d) + g_0(-2a) \right) + \left( g_2(-a-b-d) + g_1(a+b+d) + (-a-b+c-d) \right)
\]

and solving for the coefficients of \( g \) we get the augmented matrix

\[
\begin{pmatrix}
2a+b+c-d & -a & 0 & 0 \\
b+2d & a+c-d & -2a & 0 \\
-a-b-d & a+b+d & -a-b+c-d & 1
\end{pmatrix}
\]

22
which reduces to
\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\frac{2a^2}{(a^2+2a(c+d)-(b+c-d)(b-c+d))(c-d)}
\frac{2a(2a+b+c-d)}{(a^2+2a(c+d)-(b+c-d)(b-c+d))(c-d)}
\frac{2a^2+a(2b+3c-d)+(c-d)(b+c-d)}{(a^2+2a(c+d)-(b+c-d)(b-c+d))(c-d)}
\]

Having found the above coefficients for g we can now find S(k).

\[
S(k) = g(k+1) g(k) f(k)
\]

\[
= \frac{(ak^2+bk+c)(2a^2(k^2+2k+1)+(a(2b(k+1+k+3)-(2k+1))+(c-d)(b+c-d)) \prod_{j=1}^{k-1} (aj^2+bj+c)}{a^2+2a(c+d)-(b+c-d)(b-c+d))(c-d) \prod_{j=1}^{k-1} (aj^2+bj+d)}
\]

Finally we discover the closed form of this partial sum,

\[
\sum_{k=1}^{m} \prod_{j=1}^{k-1} aj^2 + bj + c = \frac{\prod_{j=1}^{k-1} aj^2 + bj + c}{\prod_{j=1}^{k+1} aj^2 + bj + d}
\]

\[
= \frac{(am^2+bm+c)(2a^2(m^2+2m+1)+a(2b(m+1)+c(2m+3)-(2m+1))+(c-d)(b+c-d)) \prod_{j=1}^{m-1} (aj^2+bj+c)}{a^2+2a(c+d)-(b+c-d)(b-c+d))(c-d) \prod_{j=1}^{m} (aj^2+bj+d)}
\]

\[
= \frac{2a^2+a(2b+3c-d)+(c-d)(b+c-d)}{(a^2+2a(c+d)-(b+c-d)(b-c+d))(a+b+d)(c-d)}
\]

23
Chapter 3: Mechanized Proofs

Combinatorial problems, such as the previous example, arise in many areas of mathematics, engineering, and theoretical physics. Their solution often entails the evaluation of a sum of products of binomial coefficients [7]. A great many distinguished mathematicians have spent an immense amount of time and ingenuity proving virtually thousands of binomial identities. We now know that much of that time and talent was wasted. Recently it has become widely known that "almost all" known binomial coefficient identities are special cases of relatively few hypergeometric identities [3]. Therefore, there are a comparatively small number of essentially different binomial identities. The problem in recognizing this has been twofold. First, the difficulty in recognizing equivalent binomial expression due to the notation used. The notation used to express binomial coefficients makes it very easy to disguise equivalent expressions in a such a way that no relationship is readily apparent. The second problem has been the reluctance of many to see the advantage of using hypergeometric series to serve as a canonical form for these identities (even though this was the method used by Euler and Gauss [7]). Although this reluctance has diminished over the last decade
and the use of hypergeometric series has increased, the need for identities has not decreased. This need for hypergeometric identities is being largely supported by the use of computers.

The increased use of hypergeometric series has directly lead to an increased use of computers in creating hypergeometric identities. Wilf asserts, "that more and more hypergeometric identities are still being conjectured and proved, so the need remains for mechanized proof" [3,78].

It would have been difficult (and in practice infeasible) to create the identity in example 2.1, without the aid of a computer. For this particular example, the software program Derive was used to find the appropriate identity. Gosper credits the program MACSYMA and the support of its developers at MIT for giving him "... the experiences necessary to provoke the conjectures that led to this algorithm" [4, 42].

Doron Zeilberger has derived other identities (using algorithms largely based on Gosper's algorithm) using the MAPLE software package. Finding identities and mathematical proofs using computers is becoming more the norm [10]. Gosper's algorithm is example of mechanized theorem proving.

Mechanized theorem proving has been a goal for a long time. The desire dates back to Leibniz in the late 17th and early 18th centuries[2].
In 1930 Herbrand proposed a system for mechanized (or automated) theorem proving based on logic rules[6]. His method was slow and untimely, since it came before the advent of the digital computer. With the arrival of the computer came a renewed interest in automated theorem proving. In the 1960's Herbrand's method was implemented on a computer by Gilmore[2]. Until 1965, "success was largely confined to problems found in logic books rather than mathematics books." [6, 5] In 1965, J.A. Robinson developed a major breakthrough. She produced the resolution inference system. This system was based on a single inference rule and did not require generalized logic axioms (only those specific to the problem). In essence the inference rule is:

an algorithm that, when successfully applied to some given set of hypotheses or premises, yields a conclusion that follows inevitably and logically from the premises. [1,197]

In other words, the algorithm is a finite automata that stops. Gosper's algorithm "infers" that a closed form can be found if a certain polynomial can be found. If this polynomial doesn't exist, Gosper's algorithm infers that no closed form can be found. Since 1965 many refinements of the resolution system have been made.

Wilf and Zeilberger have shown that a very large class of identities can be proved by computers [9]. They do not imply that computers need
to be "trusted blindly"; in fact, it is just the contrary. Although the proofs are "discovered" by a computer, all proofs presented by Wilf and Zeilberger produce a proof certificate that can be checked by hand if desired [9]. Traditionalists might complain that identity proofs generated by a computer can give no insight to the mathematician. Zeilberger cautions against being too chauvinistic, "The team human-computer is a mighty one, and an open-minded human can draw inspiration from all sources, even from a machine" [10].

I found it very enlightening to write my own program (see addendum) partially implementing Gosper's algorithm. This program will take as input the term ratio (already factored) of a specific hypergeometric series and output a partial sum or the fact that one doesn't exist. The purpose in writing this program was not to improve or advance implementations of Gosper's algorithm already in use. The purpose was strictly one of education. Perhaps most edifying were the attempts (admittedly small) at symbolic manipulation. I gained not only an insight into mechanized proofs, but an appreciation of the software techniques use in packages such as Derive.

In 1978 Gosper made a definitive step in the direction of mechanized proofs of hypergeometric series. His algorithm is now fully
implemented in MACSYMA and partially implemented in Mathematica. It will no doubt become more widely implemented when its usefulness becomes more fully recognized [8].
Appendix A

{Rick Reeves}
{1992}

Program Hyper (Input, Output);

{This program is an attempt to implement R.W. Gosper’s algorithm - perhaps not in its complete form - but perhaps in a way that gives the user an appreciation of what the algorithm entails. }

{The algorithm is not shown here due to its length. Please see Gosper’s original paper.}

USES
Gauss,  { a gaussian elimination unit - for n by n matrices }
Poly_1,  { a polynomial unit - add, subtract, multiply, show, etc.}
Crt;     { standard crt unit }

TYPE
Parameter = array [1..MAXP] of real;  { used for factored form of poly}

{-------------------------------------------------------------------}
Procedure Pause;

{ Holds user screen until key is press - then clears screen. }

VAR
Ch : Char;

Begin
GOTOXY(27,25);
Write ('Press any key to continue');
Ch := ReadKey;
CLRSCR;
end; {pause}
{---------------------------------------------------------------------}
Procedure Initialize (VAR UpperP, LowerP, P_array : Parameter);

{ This procedure initializes all the parameter arrays. }

VAR
  i : integer;

Begin
  for i := 1 to MAXP do
    Begin
      UpperP[i] := -999;
      LowerP[i] := -999;
      P_array[i] := -999;
    end; {for}

  P_array[1] := -9911; {flag - for step one}
end; {initialize}

{---------------------------------------------------------------------}
Procedure GetHyper (VAR UpperPara, LowerPara : Parameter;
  VAR UpNumber, LowNumber : Integer;
  VAR Argument : Real);

{ This procedure gets the factored hypergeometric term ratio from
  the user. Note: this maybe the same as the hypergeometric
  parameters - except for perhaps the k+1 factor in the denominator.}

VAR
  i : integer;

Begin
  GOTOXY(10,3);
  Write (OUTPUT, 'Please enter NUMBER of UPPER parameters: ');
  Readln (INPUT, UpNumber);

For i := 1 to UpNumber do
    Begin
        GOTOXY(15,5+i);
        Write (OUTPUT, 'Please enter UPPER parameter number \',i,\': \');
        Readln (INPUT, UpperPara[i]);
    end; \{for\}

GOTOXY(10,14);
Write (OUTPUT, 'Please enter NUMBER of LOWER parameters: \');
Readln (INPUT, LowNumber);

For i := 1 to LowNumber do
    Begin
        GOTOXY(15,15+i);
        Write (OUTPUT, 'Please enter LOWER parameter number \',i,\': \');
        Readln (INPUT, LowerPara[i]);
    end; \{for\}

GOTOXY(10,24);
Write (OUTPUT, 'Please enter value of argument z: \');
Readln (INPUT, Argument);

end; \{gethyper\}

{------------------------------------------------------------------}
Procedure StepOne (VAR Q_array, R_array, P_array : Parameter;
                   Up, Low : integer);

{Get p(k), q(k), and r(k) in right form, ie, put term ratio in the
  proper form. }

VAR
    i, j, p, k, x : integer;
    GOOD : boolean;
    check : real;
    Chk : integer;
Begin

\[ p := 1; \]

\{check to see if (alpha - beta) is a positive number\}

Repeat
\[
\text{GOOD} := \text{TRUE};
\]
for \( i := 1 \) to (Up) do
for \( j := 1 \) to (Low) do
begin
if \( (Q_{\text{array}}[i] <> -999.0) \) and \( (R_{\text{array}}[j] <> -999.0) \) then
begin
\[ \text{check} := Q_{\text{array}}[i] - R_{\text{array}}[j]; \]
if \( (\text{check} > 0) \) and \( (\text{int(check)} = \text{check}) \) then
begin
\[ \text{Chk} := \text{trunc(check)}; \]
\[ \text{GOOD} := \text{FALSE}; \]
\[ x := 1; \]
for \( k := p \) to \( (p + \text{Chk} - 2) \) do
begin
\[ P_{\text{array}}[k] := Q_{\text{array}}[i] - (x); \]
\[ x := x + 1; \]
end; \{for\}
\[ p := p + \text{chk} - 1; \]
\[ Q_{\text{array}}[i] := -999; \]
\[ R_{\text{array}}[j] := -999; \]
end; \{if\}
end; \{for\}
Until \text{GOOD};
for \( i := 1 \) to up do
\[ Q_{\text{array}}[i] := Q_{\text{array}}[i] + 1; \]
end; \{stepone\}
Procedure Display (Q_ay, R_ay, P_ay : Parameter;
    Up, Low    : Integer;
    Argue, Which : Real);

{ This procedure displays the polynomials in their factored form. }

VAR
    i, j : integer;

Begin
    GOTOXY (10,10);
    if which = 1.0 then
        Write ('q(k) = ')
    else
        Write ('q(k+1) = ');

    For i := 1 to Up do
        if Q_ay[i] <> -998 then
            if Q_ay[i] = 0 then
                write (' (k ) ')
            else if Q_ay[i] < 0 then
                write (' (k ,abs(Q_ay[i]):1:1,)' )
            else
                write (' (k + ,Q_ay[i]:1:1,' )
            write (' ,Argue:1:1, ) ');
        GOTOXY (10,12);
        Write ('r(k) = ');

    For i := 1 to (Low) do
        if R_ay[i] <> -999 then
            if R_ay[i] = 0 then
                write (' (k ) ')
            else if R_ay[i] < 0 then
                write (' (k ,abs(R_ay[i]):1:1,)' )
            else
                write (' (k + ,R_ay[i]:1:1, ) ');
        GOTOXY (10,14);
    i := 1;
If P_ay[1] <> -9911 then
  begin
    Write('p(k) = ');
    While P_ay[i] <> -999 do
      begin
        if P_ay[i] = 0 then
          Write('k ') 
        else if P_ay[i] < 0 then
          Write('k - abs(P_ay[i]):1:1,' ')
        else
          Write('k + P_ay[i]:1:1,' ')
        i := i + 1;
        if (WhereX > 70) then 
          Writeln;
      end; {while}
  end; {if}
else
  writeln('p(k) = 1');
  writeln;
end; {display}

{------------------------------------------------------------------}
Procedure Degree ( O_ay, R_ay, P_ay : Parameter;
  Upp, Lowp : Integer;
  Argue : Real;
  VAR P, Q, R : Poly_Type;
  VAR DegreeG, DegR,
      DegQ, DegP : Integer);

VAR
i,
DegDiff, DegSum, IntPivot : Integer;
PTerm, Qterm, Rterm, Temp, QSum, QDiff : Poly_Type;
First : Boolean;
Beta, Gamma, Pivot : Real;

{ This procedure finds the degree of the unknown function G - if it
  exists. If it doesn't exist, a message is outputted that indicates
  that. This procedure also finds the degree of all pertinent
  polynomials and puts them in standard form.
  This procedure requires units: Poly1 and Crt}
Begin
    DegR := 0;
    DegQ := 0;
    DegP := 0;
    DegSum := 0;  
    { initialize all variables to zero. }
    DegDiff := 0;
    DegreeG := 0;
    First := TRUE;
    Get (Temp);
    Temp[0] := Argue;

    { find degrees of p, q, and r }

    for i := 1 to (Low+0) do
        if (R ay[i] <> -999) then
            DegR := DegR + 1;  
            { degree of r }
        if DegR = 0 then
            R ay[1] := 1;

    for i := 1 to (Up) do
        if (Q ay[i] <> -998) then
            DegQ := DegQ + 1;  
            { degree of q }
        if DegQ = 0 then
            Q ay[1] := 1;

    for i := 1 to MAXP do
        if (P ay[i] <> -999) and (P ay[i] <> -9911) then  
            { degree of p }
            DegP := DegP + 1;

    Get (R);
    Get (Q);
    Get (P);  
    { get "zeroed" coefficient arrays }
    Get (Qterm);
    Get (Rterm);
    Get (Pterm);

    { find standard polynomial form for p, q, and r. }

    35
For $i := 1$ to $\text{Up}$ do
  if $(\text{Q}_\text{ay}[i] <> -998)$ then
    if First then
      Begin
        First := FALSE;
        Qterm[0] := Q_ay[i];
        Qterm[1] := 1;
        Multiply (Qterm,temp, Q);
        temp := Q;
      end; \{find q\}
    else
      Begin
        Qterm[0] := Q_ay[i];
        Qterm[1] := 1;
        Multiply (Qterm,temp, Q);
        temp := Q;
      end; \{else\}
    Write ('q(k+1) = '); \{display q\}
    Show(Q,DegQ);
  First := TRUE;
  Get (Temp);
  Temp[0] := 1;
For $i := 1$ to $\text{Low} + 0$ do
  if $(\text{R}_\text{ay}[i] <> -999)$ then
    if First then
      Begin
        First := FALSE; \{find r\}
        Rterm[0] := R_ay[i];
        Rterm[1] := 1;
        Multiply (Rterm,temp, R);
        temp := R;
      end; \{if\}
    else
      Begin
        Rterm[0] := R_ay[i];
        Rterm[1] := 1;
        Multiply (Rterm,temp, R);
        temp := R;
      end; \{else\}
Write ('r(k) = '); {display r}
Show(R,DegR);
First := TRUE;
Get (Temp);
Temp[0] := 1;
For i := 1 to MAXP do
  if (P_ay[i] <> -999) and (P_ay[i] <> -9911) then
    if First then
      Begin
        First := FALSE;
Pterm[0] := P_ay[i];
Pterm[1] := 1;
Multiply (Pterm,temp, P); { find p}
temp := P;
      end; {if}
    else
      Begin
        Pterm[0] := P_ay[i];
Pterm[1] := 1;
Multiply (Pterm,temp, P);
temp := P;
      end; {else}
  Write ('p(k) = ');
if DegP > 0 then
  Show(P,DegP) {display p}
else
  begin
    Writeln ('1');
P[0] := 1;
  end;
Pause;
{ find term sums and differences }
Add (R,Q,QSum);
For i := 0 to MAXP do
  if QSum[i] <> 0 then
    DegSum := i;
Write ('Qsum(k) = ');
Show (QSum, DegSum);
Subtract (Q,R,QDiff);
For i := 0 to MAXP do
   if QDiff[i] <> 0 then
      DegDiff := i;
   Write ('QDiff(k) = ');
   Show (QDiff, DegDiff);
   {Will the degree of our mystery polynomial stand up and sign in please.}

   Pivot := 0;
   Beta := QDiff[DegDiff];
   Gamma := QSum[DegSum];
   if Gamma <> 0 then
      Pivot := -2 * beta / gamma;
   IntPivot := trunc(Pivot);
   if DegDiff >= DegSum then
      DegreeG := DegP - DegDiff
   else if (Pivot > (degP - DegSum + 1)) and (Pivot = IntPivot) then
      DegreeG := IntPivot
   else
      DegreeG := (degP - DegSum + 1);
   Writeln ('Degree of G = ',DegreeG);
   DegreeG := 2;
end; {degree}

{-----------------------------------------------}
Function Factorial (n : integer): integer;

   { This is a recursive function that finds factorials.}

begin
   if n = 0 then
      Factorial := 1
   else
      Factorial := n * Factorial(n-1)
end; {factorial}
Function Combo (n,k :integer): integer;

   { This function finds combinations - it uses function factorial.}
Var
temp : integer;

begin
   if n <= k then
      temp := 1
   else
      temp := Factorial(n) Div (Factorial(k) * Factorial(n-k));
   Combo := temp;
end; {combo}

Procedure BuildSK(Var Mat : Matrix_Type; Degree : Integer);

   { This procedure finds the term (in matrix form) g(k-1) }

Var
   i, j : integer;

Begin
   For i := 1 to Degree+1 do
      For j := i to Degree+1 do
         if odd (j-i) then
            Mat[i,j] := Combo(j-1,j-i)*(-1)
         else
            Mat[i,j] := Combo(j-1,j-i) ;
   For i := 1 to Degree+1 do
      begin
         For j := 1 to Degree+1 do
            Write (Mat[i,j]:8:1);
         writeln;
      end;
   Pause;
end; {buildSK}
Procedure BuildS (Var Mat : Matrix_Type; Degree : Integer);

{ This procedure finds the term (in matrix form) g(k) }

Var
  i, j : integer;

Begin
  For i := 1 to Degree + 1 do
    Mat [i,i] := 1;
  For i := 1 to Degree + 1 do
    begin
      For j := 1 to Degree + 1 do
        Write (Mat [i,j]:8:1);
        writeln;
      end;
    Pause;
end; {buildS}

{ -------------------------------------------------------------- }
Procedure BuildTerm (Mat : Matrix_Type; Poly : Poly_Type;
  DegM,DegP : Integer; Var Term : Matrix_type);

{ This procedure is used to find the terms (in matrix form) q(k+1) g(k)
  and r(k) g(k-1). }

Var
  i, j, k : integer;

Begin
  For i := 0 to DegP do
    writeln (poly[i]:3:0);
  For i := 0 to DegP do
    For j := 1 to DegM + 1 do
      For k := 1 to DegM + 1 do
        Term[i+j,k] := Term[i+j,k] + (Poly[i] * Mat[j,k]);
For i := 1 to DegM+1+DegP do
  Begin
    For j := 1 to DegM+1+DegP do
      Write (Term [i,j]:8:1);
      writeln;
    end;
  Pause;
end;  {Term}

{-----------------------------------------------}
Procedure Subtract (Mat1, Mat2 : Matrix_type; Var Answer : Matrix_Type);

{ This procedure is used to subtract terms that are in matrix form }

Var
  i, j  : integer;

begin
  for i := 1 to MAXMATRIX do
    for j := 1 to MAXMATRIX do
      Answer[i,j] := Mat1[i,j] - Mat2[i,j];
  end;

{-----------------------OUTPUT---------------------}

Procedure OUTPUT(ANSWER : Answer_Type; B : Matrix_Type; IERR,N : integer);

{ This procedure outputs to a file (Defaulted to Screen for now) the
  Gaussian reduced matrix and the solution set - or a message that
  no unique solution exits.}

var
  Outfile : text;   {allows programmer to pick output device}
  I, J  : Integer;  {loop and matrix counters}

begin
  assign(Outfile, 'CON');
  rewrite(Outfile);
{ If there are not errors output the reduced matrix and the solution set. }

if IERR = 0 then
  begin
    writeln(Outfile);
    writeln(Outfile);
    writeln(Outfile,'The Gaussian reduced matrix is as follows:');
    writeln(Outfile);
    for I := 1 to N do
      begin
        for J := 1 to (N + 1) do
          write(OUTFILE, B[I,J] :12:2);
        writeln(OUTFILE);
      end; {for}
    writeln(Outfile);
    writeln(Outfile,'The solution set is as follows:');
    writeln(Outfile);
    for I := 1 to N do
      writeln(Outfile,'G',I,' = ', ANSWER[I]:12:2);
    writeln(Outfile);
  end {if}

{ Else, output that there are no unique solutions to this linear system. }

else
  begin
    writeln(Outfile);
    writeln(Outfile);
    writeln (Outfile,'There is no unique solution to this linear system.');
    writeln(Outfile);
    for I := 1 to N do
      begin
        for J := 1 to (N + 1) do
          write(OUTFILE, B[I,J] :8:1);
        writeln(OUTFILE);
      end; {for}
    writeln(Outfile);
  end; {else}
close(OUTFILE);
Pause;
end; {output}
Procedure FindClosed (PolyG, P,Q,R : Poly_Type; DegG,DegR,DegQ,DegP : Integer);

{ This procedure is used to find the closed form of the summation.}

Var
Temp : Poly_Type;

begin
Get(Temp);
Multiply (Q,PolyG,Temp);
Multiply (Temp,R,Temp);
Show(temp, DegG + DegQ + DegR);
Pause;
end; {findclosed}

Procedure FindPoly (P,Q,R : Poly_Type; DegG,DegR,DegQ,DegP : Integer);

{ This procedure finds the coefficients of G (if they exist).}

Var
SK, SK_one : Matrix_Type;
Answer,
FirstTerm,
SecondTerm : Matrix_Type;
i,j, N, Error : Integer;
Coef : Answer_Type;
Max : Integer;
PolyG : Poly_Type;

Begin
error := 0;

Initialize_Matrix (SK);
Initialize_Matrix (SK_one);
Initialize_Matrix (FirstTerm);
Initialize_Matrix (SecondTerm);
Initialize_Matrix (Answer);
Get (PolyG);
BuildS (SK,DegG);
BuildSK (SK_one, DegG);
BuildTerm(SK, Q, DegG, DegQ, FirstTerm);
BuildTerm(SK_one , R, DegG, DegR, SecondTerm);

Subtract (FirstTerm, SecondTerm, Answer);

if DegQ > DegR then 
    N := DegQ 
else 
    N := DegR;
N := N + DegG + 1;

if DegG+2 > N Then 
    Max := DegG+2 
else 
    Max := N;

For i := 1 to MAXMATRIX do
    Answer[i, DegG+2] := P[i-1];

For i := 1 to MAX do 
    begin 
    for j := 1 to Max +1 do 
        write (Answer[i,j]:8:1); 
        writeln; 
    end; 
end; 

Pause; 

N := DegG+1; 
Gaussian_Elim (Answer,ERROR,N , Coef); 
Output ('Coef, Answer,ERROR, N);
For i := 1 to DegG+1 do 
    PolyG[i-1] := Coef[i];
write ('g(k) := ');
show(PolyG, DegG);
Pause;
FindClosed (PolyG, P,Q,R,DegG,DegR,DegQ,DegP);
end; 

{ FindS }
VAR
    Upper,
    Lower,
    P_array : Parameter;
    UpNum,     {number of upper parameters}
    LowNum,    {number of lower parameters}
    Argue : Real;
    DegG,DegR,
    DegQ,DegP : Integer;
    P, Q, R : Poly_Type;

Begin
    CLRSCR;
    Initialize (Upper, Lower, P_array);
    GetHyper (Upper, Lower, UpNum, LowNum, Argue);
    Pause;
    Display (Upper, Lower, P_array, UpNum, LowNum, Argue,1);
    Pause;
    StepOne (Upper, Lower, P_array, UpNum, LowNum);
    Display (Upper, Lower, P_array, UpNum, LowNum, Argue,2);
    Degree (Upper, Lower, P_array, UpNum, LowNum, Argue,P,Q,R,DegG,DegR,DegQ,DegP);
    Pause;
    if DegG >= 0 then
        begin
            FindPoly (P,Q,R,DegG,DegR,DegQ,DegP);
        end {if}
    else
        begin
            writeln (' not summable with respect to hypergeometric terms ');
            writeln (' i.e., sum t(k) is not a hypergeometric term');
        end {else}
end. {hyper}
{Rick Reeves}

Unit Gauss;

{ This unit performs gaussian elimination on n by n matrices. }

**************************************************************************

INTERFACE

Uses

CRT;

Const

MAXMATRIX = 12;
MAXANSWER = 20;

Type

Matrix_Type = Array [1..MAXMATRIX, 1..MAXMATRIX] of real;
Answer_Type = Array [1..MAXANSWER] of real;

Procedure GAUSSIAN_ELIM(var B : Matrix_Type; var IERR,N : integer;
                          var Answer : Answer_Type);

Procedure Initialize_Matrix(VAR Mat : Matrix_Type);

**************************************************************************

IMPLEMENTATION

VAR

ZERO_MATRIX : Matrix_Type;

Procedure GAUSSIAN_ELIM(Var B : Matrix_Type; Var IERR,N : integer;
                        Var Answer : Answer_Type);

{ This procedure determines whether a linear system ( in the form of
  a augmented matrix) has a unique solution.

  B     Augmented matrix from the driver.
IERR  Error code passed from the driver (See driver variables above).
N    Number of equations in system.
P    The smallest integer in the column that needs to have its
     elements eliminated.
H,
K,
and L  These are used to manipulate the various elements of the matrix.
TEMP  This variable is used in the switching of rows - if the smallest
     P is not in the same row as I.
MJI   This is the multiple used for creating zeros in the Gaussian
     eliminations.
TOTAL This variable is used to determine the correct solutions during
     backward substitution.
I,
J    Both used as row and column indices.
ANSWER This array will hold the solution set - if one exits. }

var
  P,H,K,L,I,J : integer;
  TEMP, MJI, TOTAL : real;

begin
  IERR := 0;
  I := 1;

  { If there are no errors, repeat the following for 1 through (N-1) }

  While (IERR = 0) and (I < = (N-1) ) do
  begin

    { Find the smallest integer P (no greater than N) for the particular
     column search. }

    P := I;
    While (abs(B[P,I]) = 0.0) and (P < = N ) do
    begin
      P := P + 1;
  end

  end

47
{ If no such P can be found then there is no unique solution. }

    if P = (N + 1) then
        IERR := 1

{ If the value, P, found was not in the top row, I, then switch rows:
  Ep Swithed with Ei . }

else
    begin
      if ( P <> I) then
        begin
          for H := 1 to (N + 1) do
            begin
              Temp := B[I,H];
              B[P,H] := TEMP;
            end; {for}
        end; {if}

{ Perform the Gaussian Elimination on the elements below the chosen I,J
  element. }

    H := I + 1;

{ Step 1 - find the multiples that will eliminate the other terms in
  in the particular column. }

    for J := (I+1) to N do
      begin
        MJI := B[J,I] / B[I,I];
{ Step 2 - Perform \((E_j - M_j*E_i)\) and switch with \(E_j\). }

\[
\text{for } K := H \text{ to } (N + 1) \text{ do}
\begin{align*}
B[J,I] & := 0.0;
\end{align*}
\text{end; } \{\text{for}\}
\]
\text{end; } \{\text{else}\}
\begin{align*}
I & := I + 1; \\
\text{end; } \{\text{while}\}
\end{align*}
\text{if } IERR = 0 \text{ then}
\begin{align*}
\text{begin}
\end{align*}
\{ If the Nth element of the last row equals zero - there is no unique element.\}
\begin{align*}
\text{if } (\text{abs}(B[N,N]) = 0.0) \text{ then}
\begin{align*}
IERR & := 1
\end{align*}
\end{align*}
\{ If not then perform backward substitution.\}
\begin{align*}
\text{else}
\begin{align*}
\text{begin}
\end{align*}
\{ Step 1 - Solve for the single variable of the last row first. \}
\begin{align*}
\text{ANSWER[N]} & := B[N,(N + 1)] / B[N,N];
\end{align*}
\{ Step 2 - Using the above answer (and all previous answers), solve the next variable. \}
\begin{align*}
\text{for } K := 1 \text{ to } (N - 1) \text{ do}
\begin{align*}
\text{begin}
I & := (N - 1) - K + 1; \\
H & := I + 1; \\
\text{TOTAL} & := B[I,(N + 1)];
\end{align*}
\text{for } L := H \text{ to } N \text{ do}
\begin{align*}
\text{TOTAL} & := \text{TOTAL} - B[I,L] * \text{ANSWER[L]}; \\
\text{ANSWER[I]} & := \text{TOTAL} / B[I,I];
\end{align*}
\text{end; } \{\text{for}\}
\text{end; } \{\text{else}\}
\text{end; } \{\text{if}\}
\text{end; } \{\text{Gaussian}\}
\]
Procedure Initialize_Matrix (VAR Mat: matrix_type);
begin
  Mat := ZERO MATRIX;
end; {initialize}

VAR
  i, j : integer;

Begin  {Initialization section}
  For i := 1 to MAXMATRIX do
    For j := 1 to MAXMATRIX do
      ZERO MATRIX [i,j] := 0.0;

end. {Gauss unit}
UNIT Poly_1;

{This unit is a polynomial machine. See below for data structure}

{******************************************************************************}

INTERFACE

CONST
  MAXP = 70;

TYPE

  Poly_Type = ARRAY [0..MAXP] of Real;

Procedure Get (VAR poly_out : Poly_Type);
Procedure Show (poly_in : Poly_Type; Deg : integer);
Procedure Add (first_in, second_in:Poly_Type; VAR answer_out:Poly_Type);
Procedure Subtract (first_in, second_in:Poly_Type; VAR answer_out:Poly_Type);

Procedure Multiply (first_in, second_in:Poly_Type;VAR answer_out:Poly_Type);
Function Evaluate (poly_in : Poly_Type; x_value: REAL): REAL;

{******************************************************************************}

IMPLEMENTATION

USES
  Crt;

VAR
  ZERO_POLY : Poly_Type;

51
Procedure Get (VAR poly_out : Poly_Type);

Begin
    Poly_Out := ZERO_POLY;

end; {show}

Procedure Show (poly_in : Poly_Type; Deg : integer);

VAR
    x, y, i : integer;

Begin
    writeln;
    writeln;
    y := WhereY;
    For i := 0 to Deg do
        Begin
            if WhereX > 70 then
                begin
                    writeln;
                    writeln;
                    writeln;
                    writeln;
                end;
            write (poly_in[deg-i];5:2);
            if deg - i <> 0 then
                write('k');
            x := WhereX;
            GOTOXY (x+1, y-1);
            if deg - i <> 0 then
                Begin
                    write (deg-i);
                    GOTOXY (x+5, y);
                    write ('+');
                end; {if}
        end; {for}
    writeln;

end;
writeln;
writeln;
end; {show}

{--------------------------------------------------------------------------}
Procedure Multiply ( First_in, Second_in : Poly_Type;
VAR Answer_out : Poly_Type);

VAR
i,j : integer;

Begin
Answer_out := ZERO_POLY;
For i := 0 to MAXP do
  For j := 0 to MAXP do
    Answer_out[i+j] := Answer_out[i+j] +
    ( First_in[i] * Second_in[j]);
end;

{--------------------------------------------------------------------------}
Procedure Add ( First_in, Second_in : Poly_Type;
VAR Answer_out : Poly_Type);

VAR
i : integer;

Begin
Answer_out := ZERO_POLY;
For i := 0 to MAXP do
  Answer_out[i] := First_in[i] + Second_in[i];
end;
Procedure Subtract (First_in, Second_in : Poly_Type;
           VAR Answer_out : Poly_Type);

VAR
i : integer;

Begin
  Answer_out := ZERO_POLY;
  For i := 0 to MAXP do
    Answer_out[i] := First_in[i] - Second_in[i];
end;

Function Evaluate (poly_in : Poly_Type; x_value: REAL): REAL;

Begin

end;

VAR
i : BYTE;

Begin  {Initialization section}
  For i := 0 to MAXP do
    ZERO_POLY[i] := 0.0;
end.

54
References


