APL and Games

Algorithms for Concepts in the Theory of Cooperative Games

Joachim Rosenmüller

1Institute of Mathematical Economics
IMW
University of Bielefeld
D–33615 Bielefeld
Germany

May 31, 2011
We present a coarse introduction into “A Programming Language” streamlined as to lead to Programs for Cooperative Games and their solution concepts.
## Contents

1 Preliminaries .......................................................... 1
  1 Basics ........................................................................... 2

2 Arrays ............................................................................. 5
  1 Definitions and “*shape*” ............................................. 7

3 Functions ........................................................................ 13
  5 Encode and Decode ..................................................... 14
  8 Grade Up and Grade Down ........................................... 20
  9 The Dyadic Transposition $\&$ ......................................... 23

4 Operators ......................................................................... 29
  1 Ravel ............................................................................ 30
  2 Outer Product ............................................................. 33
  3 Inner Product ............................................................ 37

5 The definition of functions: Programming ....................... 41
  1 The Definition of Functions – Programming in APL ........ 42

6 The Coalitional function .................................................. 47
  1 Games .......................................................................... 48

7 Cephoids ......................................................................... 55
# Contents

1. Faces ................................ 56
2. Duality ................................ 62
3. 1\textsuperscript{st} Algorithm ............... 66
4. 2\textsuperscript{nd} Algorithm ............... 76
5. 3\textsuperscript{rd} Algorithm ............... 78

Bibliography ................................ 91
Chapter 1

Preliminaries

This chapter constitutes a coarse introduction. Basically we assume that the reader is familiar with some basic procedures or operations as offered in any APL textbook. Thus, this chapter does not serve as a foundation for the beginner; we claim neither completeness nor self evidence.
1 Basics

We do not really want to give an introductory course to APL. Therefore, these preliminaries consist of some rough notes concerning the basic operations or functions. It is hence tacitly assumed that the reader is informed about basic features of APL that can be acquired from any textbook and via trial and error.

APL is a (very condensed) Programming Language. It is based on a set of symbols/fonts that reflect functions or operators as well as the arguments of such functions in a wide sense. Here is the APL keyboard.

Elementary algebraic operation look quite familiar on first sight. We have (the program answering on the left margin):

```
3+7
10
5-11
-6
```
5 ÷ 2
2.5
6 × 6
36
2*5
32
9*(1÷2)
3
4*(- 1÷2)
0.5

However, it is advisable to regard operations like * as functions on reals or integers, cartesian products of reals, etc. E.g., * : R × R → R suggests an obvious function in 2 variables; of course the computer does not know “reals”, but for our purpose it is convenient not to emphasize this deficiency.
Chapter 2

Arrays

Within this chapter we discuss arrays. Basically, an array is a variable. Intuitively, an array is a higher dimensional matrix; slightly more formally a function from (the Cartesian product of) finitely many index sets into the reals. (Yes, we recall that there are no reals in the computer...). An index set is typically of the form

$$I = [1, n] := \{1, \ldots, n\};$$

for some $n \in \mathbb{N}$ which we conveniently call an “interval”. In other words, we are just counting the rows, columns, sheaves, .... of $K$–dimensional matrix by means of appropriate intervals.

Here a problem may occur as we sometimes want to represent functions on the Cartesian product of arbitrary finite index sets. As APL does not know such generalized matrices, we have to take care that a suitable bijection is introduced which allows to treat our non–interval–like index set via some suitable interval. For the user, this niceties may not be relevant, it is clear to him how a generalized matrix has to look like. For Mathematical precision one may think otherwise.

Arrays are the main substrate of APL. They can easily be introduced by assigning values to the various entries via the “shape” operation. They can easily be handled, i.e., extended, mutilated, composed, or partially adjusted by suitable operations. They have the usual structure assigned to them by elementary linear algebra, like we may add them, multiply them with a constant, etc.. Also, there are several very general versions of a product of two arrays.

Moreover, turning to functions more involved, arrays serve as arguments of
such functions. We will be able to program whatever a “function” could be and, as rule, it will be a function defined on a certain class of arrays.

Also, as arrays may have numerical or literal entries, we can manipulate text variables that are certain arrays and convey messages, instructions, explanations, headers, ..... etc. during the process of programming or computing via a program.
1 Definitions and “shape”

We consider “intervals”

\[ I = [1, n] := \{1, \ldots, n\} \quad (n \in \mathbb{N}) ; \]

in addition we shall use the interval

\[ I = [1, 0] := \emptyset \]

in order to indicate the empty set.

**Definition 1.1.** Let \( I_1, \ldots, I_K \) be intervals. An array (of degree \( K \)) is a mapping

\[ A : I_1 \times \ldots \times I_K \to \mathbb{R} . \]

For convenience, the case \( K = 0 \) is included. An array

\[ A : \prod_{L=1}^{0} I_L \to \mathbb{R} \]

is by convention a **real number**.

Note that \( \mathbb{AP} \) distinguishes between the real number \( a \) and the one dimensional vector \( (a) \). The first object is given by the (mapping, function, array)

\[ A : \prod_{L=1}^{0} I_L \to \mathbb{R} , \quad A[\cdot] = a , \]

while the second one is

\[ A : \prod_{L=1}^{1} I_L \to \mathbb{R} , \quad A[1] = a . \]

In particular, for \( K = 1 \), the array

\[ \theta := [1, 0] \to \mathbb{R} \]

is an array of degree 1 called **zilde**.
Remark 1.2. The range of an array $A$ can be the set $ALPHABET := \{CHARACTERS\}$ instead of $\mathbb{R}$. That is, the above definition should be augmented by an analogous definition of

$$C : \prod_{L=1}^{K} I_L \rightarrow ALPHABET.$$ 

Definition 1.3. we shall use the following notation:

$$A^K := \{A | A \text{ is a (numerical) array of degree } K, \}$$

$$C^K := \{C | C \text{ is a (character) array of degree } K.\}$$

Remark 1.4. 1. Note that the elements of $A^0, A^1, C^0,$ and $C^1$ are the only ones that $APL$ accepts directly via the keyboard; we may use “assignment” $\leftarrow$ to simultaneously create an array $A \in A^0$ and attach a value to it. Similarly, we assign the values of a vector by using $\leftarrow$ and typing the vector as a string of numbers with empty spaces in between. This “geometrical” representation of a vector is quite in accordance with the everyday usage in Mathematics $a = (a_1, a_2, a_3, \ldots, a_n)$. However, in order to enter arrays of higher degrees, an appropriate procedure will be necessary.

2. Real numbers and characters are arrays of degree 0, hence there is a formal identification or “imbedding” indicated by

$$A^0 = \mathbb{R}, \ C^0 = ALPHABET$$

3. Naturally, arrays of degree 2 are called matrices. While we regard them as mappings

$$A := I_1 \times I_2 \rightarrow \mathbb{R}$$

or

$$C := I_1 \times I_2 \rightarrow ALPHABET,$$

the computer (again in accordance with everyday Mathematics usage) writes them as two dimensional arrays. The procedure to induce the machine doing so will be presented immediately.

Example 1.5.

```
A + 3 5 5 55 7 8 1 1.666 (1 - 3)
A
3 5 5 55 7 8 1 1.666 0.3333333333
A + 3
```
Definition 1.6 (The monadic \( \rho \) “reshape”). We define the monadic operator \( \rho \) as follows:
\[
\rho : A^K \rightarrow A^1
\]
\[
\rho(A) := (n_1, \ldots, n_K) = (|I_1|, \ldots, |I_K|)
\]
That is, \( \rho(A) \) is an array of degree 1 or formally
\[
\rho(A) : [1, \ldots, K] \rightarrow \mathbb{R}
\]
the coordinates being \( \rho[L] = n_L \) for \( L = 1, \ldots, K \). In particular, if \( A \in A^0 \) then we find \( \rho A : [1,0] \rightarrow \mathbb{R} \), thus \( \rho A = \theta \) and similarly for \( A \in C^0 \).

Remark 1.7.  
1. For every array \( A \) we have \( \rho \rho A = (K) \in A^1 \).
2. The arrays \( 4 \in A^0 \) and \( (4) \in A^1 \) are well distinguished, as a consequence we observe \( \rho 4 = \theta \) but \( \rho (4) = (1) \). Both the latter quantities are vectors. Therefore we continue with \( \rho \rho 4 = (0), \rho \rho (4) = (1) \), etc.

Example 1.8.

\[
\begin{align*}
A+4 & 5 5 5 5 8 7 1 (1+3) 5.6 \\
rA & \\
9 & \\
\rho rA & \\
1 & \\
\rho \rho rA & \\
1 & \\
************** & \\
\rho 4 & (ZILDE, i.e. \( \theta \)) \\
\rho \rho 4 & \\
0 & ZILDE is a vector defined on \([1,0]\), hence \( \rho \) returns the lengthvector of this interval which \( i (0) \). Clearly a further application
\end{align*}
\]
Now we observe that APL knows dyadic operators as well. These are functions defined on pairs of operators which may have to obey certain restrictions. We start out with the dyadic version of the operator $\rho$ that we have just studied.

**Definition 1.9** (The dyadic $\rho$ “**reshape**”). 1. First let is introduce the space of arrays which have values in the set of integers, i.e., the inte-
gervaled generalised matrices. Formally, we introduce

\[ N^K := \left\{ A \in \mathcal{A}^K \mid A : \prod_{L=1}^{K} I_L \to \mathbb{N}_0 \right\} . \]

In particular, \( N^1 \) denotes integer vectors.

2. The **dyadic** \( \rho \) is defined via

\[ \rho : N^1 \times A^1 \to \bigcup_{K=0}^{\infty} A^K \]

as follows:

(a) \( (n_1, \ldots, n_K)\rho(x_1, \ldots, x_r) = B \in \mathcal{A}^K \)

(b) If \( n_L \geq 1 \) (\( L = 1, \ldots, K \)) and \( r \geq \prod_{L=1}^{K} n_L \) then

\[ B(1,1,\ldots,1) = x_1, \]
\[ B(1,1,\ldots,1,2) = x_2, \]

etc., lexicographically, i.e.

\[ B(1,\ldots,1,\bullet) = (x_1,\ldots,x_{n_K}) \]
\[ B(1,\ldots,2,\bullet) = (x_{n_K+1},\ldots,x_{2n_K}) \]

etc.

(c) If \( n_L = 0 \) for some \( L \in \{1, \ldots, K\} \), then \( B \) “has no values” – but it is well defined. In other words, one dimension is 0 and hence \( B \) is a “flat” array – a relative of ZILDE. As we expect (and will see immediately), ZILDE can be seen as a canonical member of this family, as \( \Theta = 0\rho 1 \) can be verified.

(d) If \( r > \prod_{L=1}^{K} \) is true, then the remaining values of \( x \) are omitted.

(e) If \( r < \prod_{L=1}^{K} \) is true, then the values of \( x \) are repeated. I.e.,

\[ n\rho x = n\rho(x,x,\ldots,x) \]

holds true.

**Remark 1.10.** The following details are consequences from our definition:
1. If $x$ is a vector and $n$ a natural number, then $\rho_n x = n$.

2. $(1)\rho(x_1, \ldots, x_r) = (x_1), \quad \Theta \rho(x_1, \ldots, x_r) = x_1$, as the last object has the dimension as indicated in the first argument of the dyadic $\rho$, hence it has to be a real number.

   Indeed we find in addition that $0\rho(x_1, \ldots, x_n) = \Theta$; hence, $\rho$ is actually defined on $N_0 \times A_1$.

3. Alphabet–valued arrays $B \in \mathbb{C}^K$ are generated in quite the same way
Chapter 3

Functions

Functions are generally defined on (Cartesian products of) certain spaces of arrays. Some of these we have implicitly been dealing with; any manipulation of an array is, in a sense, of course a function. Furthermore, there are quite a few functions built into the APL language, these are called “system functions”. Such functions may be elementary (e.g., addition or multiplication) or more involved (see the following examples, e.g., encode and decode or the dyadic transposition seen as a permutation acting on vectors, We will present a few of them from our more formal viewpoint but, again, no claim regarding completeness will be maintained.

Later on, however, we would like to generate additional functions for any particular purpose we are interested in. This technique is what in everyday language is called “Programming”. We postpone this presentation to Chapter 5.
5 Encode and Decode

**Remark 5.1.** Let \( b \in \mathbb{N} \) represent the “basis of \( b \)-adic representation” and let \( x \in \mathbb{N}_0 \) be an arbitrary natural number including 0. Then there exists uniquely \( n \in \mathbb{N} \) minimally chosen and an integer vector \( x = (x_0, \ldots, x_n) \in [0, b-1]^n \) such that

\[
x = x_0 + x_1b + x_2b^2 + \ldots + x_nb^n
\]

holds true. \( x = (x_0, \ldots, x_n) \in [0, b-1]^n \) is called the **\( b \)-adic representation of** \( x \).

Another way of representing \( x \) in terms of the basis \( b \) is indicating by the notation

\[
x = x_0 + b(x_1 + x_2b + \ldots x_nb^{n-1}) = x_0 + b(x_1 + b(x_2 + \ldots x_nb^{n-1})) \]

\[
\vdots
\]

\[
= x_0 + b(x_1 + b(x_2 + \ldots b(x_n-1 + bx_n) \ldots)).
\]

Now, the term in parenthesis is the largest multiple of \( b \) contained in \( x \). Thus, using the Gauss–bracket we obtain

\[
x_0 = x - \left[ \frac{x}{b} \right] b
\]
\[
x_1 = x_0 - \left[ \frac{x-x_0}{b} \right] b
\]
\[
\vdots
\]
\[
x_n = x_0 - \ldots - x_{n-1}b^{n-1} - \left[ \frac{x-x_0 - \ldots - x_{n-1}b^{n-1}}{b} \right] b.
\]

In other words, if we define successively “quotients” \( q_1, q_2, \ldots, q_n \) via

\[
x = \left[ \frac{x}{b} \right] b + \bar{x}_0 := q_1b + \bar{x}_0
\]
\[
q_1 = \left[ \frac{q_1}{b} \right] b + \bar{x}_1 := q_2b + \bar{x}_1
\]
\[
\vdots
\]
\[
q_{n-1} = \left[ \frac{q_{n-1}}{b} \right] b + \bar{x}_{n-1} := q_nb + \bar{x}_{n-1}
\]
\[
q_n < b, q_n := \bar{x}_n.
\]
then it is clear that 

\[(x_0, \ldots, x_n) = (\bar{x}_0, \ldots, \bar{x}_n)\]

is the case. This way we see that the coefficients \(x_i\) are obtained by successively “dividing with remainder”. This procedure is known as the Euclidean Algorithm.

There is a more general version of representing integers with respect to a basis which we describe shortly as follows.

Let \(b = (b_1, \ldots, b_{n+1}) \in \mathbb{N}^{n+1}\) be an integer vector and let \(x \in \mathbb{N}\) be such that

\[x \leq b_1 + b_2 b_1 + \ldots + b_2 \cdots b_{n+1}\]

holds true. Then there exists uniquely \((x_0, \ldots x_n) \in \mathbb{N}^{n+1}\) satisfying

\[0 \leq x_i < b_{i+1} \quad (i = 0, \ldots, n)\]

and

\[x = x_0 + x_1 b_1 + x_2 b_1 b_2 + \ldots x_n b_1 b_2 \cdots b_n = x_0 + b_1(x_1 + b_2(x_2 + \ldots b_1(x_{n-1} + b_n x_n)\ldots)).\]

Again, the coefficients \((x_0, \ldots, x_n)\) are obtained by the (slightly modified) Euclidean Algorithm. In the present context this is expressed via

\[
x = \left[ \frac{x}{b_1} \right] b_1 + \bar{x}_0 := q_1 b_1 + \bar{x}_0
\]

\[
q_1 = \left[ \frac{q_1}{b_2} \right] b_2 + \bar{x}_1 := q_2 b_2 + \bar{x}_1
\]

\[
\vdots
\]

\[
q_{n-1} = \left[ \frac{q_{n-1}}{b_n} \right] b_n + \bar{x}_{n-1} := q_n b_n + \bar{x}_{n-1}
\]

\[
q_n < b_{n+1}, q_n := \bar{x}_n
\]

with \(0 \leq x_i < b_{i+1}\) for \(i = 0, \ldots, n\).

Now this is formalized suitably for the APL -context as follows.

**Definition 5.2** (Encode \(\tau\)). We define

\[\tau : \mathbb{N} \times \mathbb{N}_0 \to \mathbb{N}_0\]

\[(b_{n+1}, \ldots, b_1) \tau x := (x_n, x_{n-1}, \ldots, x_0)\]

such that
\[
(8) \quad x = x_0 + x_1 b_1 + x_2 b_1 b_2 + \ldots x_n b_1 b_2 \cdots b_n \\
= x_0 + b_1(x_1 + b_2(x_2 + \ldots b_1(x_{n-1} + b_n x_n)\ldots)).
\]

with
\[
(9) \quad 0 \leq x_i < b_{i+1} \quad (i = 0, \ldots, n)
\]
is satisfied.

**Example 5.3.** If the basis consists of the powers of an integer, then we obtain the first examples of dyadic or decimal expansion:

\[
\begin{align*}
2 & 2 2 2 \tau 8 \\
1 & 0 0 0 \\
10 & 10 10 10 10 \tau 456 \\
0 & 0 4 5 6
\end{align*}
\]

Similarly, hexagesimal expansion can be generated as follows:

\[
\begin{align*}
\text{B} & \leftarrow '0123456789\text{ABCDEF}' \\
& \leftarrow '0123456789\text{ABCDEF}'[1 + (2\cdot16)\tau 19] \\
13 & \leftarrow '0123456789\text{ABCDEF}'[1 + (2\cdot16)\tau 15] \\
0F & \leftarrow '0123456789\text{ABCDEF}'[1 + (2\cdot16)\tau 256] \\
00 & \leftarrow '0123456789\text{ABCDEF}'[1 + (3\cdot16)\tau 256] \\
100 & \leftarrow '0123456789\text{ABCDEF}'[1 + (4\cdot16)\tau 256]
\end{align*}
\]

Here are some further conventions that extend the range of \(\tau\):

If \(x > b_{-1}\) is the case, then \(\tau\) yields the result of the Euclidean algorithm up to \(n\), that is, terminates at \(n\):
Also, there is an additional convention for 0 being introduced into the basis: the algorithm stops at $n$ but yields the remainder. For example, we find

\[
\begin{align*}
\text{N+10} \\
(Np2) \tau N-1 \\
0 0 0 0 0 1 0 0 1 \\
(Np10) \tau N-1 \\
0 0 0 0 0 0 0 0 9 \\
(Np10) \tau N+1 \\
0 0 0 0 0 0 0 1 1 \\
(Np10) \tau 2N \\
0 0 0 0 0 1 0 2 4 \\
(Np2) \tau 2N \\
0 0 0 0 0 0 0 0 0
\end{align*}
\]

\[
\begin{align*}
((0, Np2)) \tau 2N \\
1024 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
\end{align*}
\]

\[
\begin{align*}
((0, Np2)) \tau 1+2N \\
1024 0 0 0 0 0 0 0 0 0 1
\end{align*}
\]

\[
\begin{align*}
((4, 6, 7, 0), Np2) \tau (2N+1) \\
0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 \\
((4, 6, 7, 0), Np2) \tau 1 + 2N \\
0 0 0 1 0 2 4 0 0 0 0 0 0 0 0 0 0 0 0 1
\end{align*}
\]

\[
\begin{align*}
((55, 55, 55, 0), Np2) \tau 1 + 2N \\
0 0 0 10 2 4 0 0 0 0 0 0 0 0 0 1
\end{align*}
\]
That is, if the 0 appears at some instant, then $\tau$ yields the remainder of the Euklidean algorithm and ignores all further data. This way $\tau$ has been extended as to be a mapping

$$\tau : \mathbb{N}_0^n \times \mathbb{N}_0 \rightarrow \mathbb{N}_0^n$$

Moreover, the action on vectors and matrices is defined coordinate wise: e.g., if we start out with

$$V \leftarrow 1 + 12 \times 4$$

then we obtain

\[
\begin{array}{ccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

and as $\tau$ lists the dyadic expansion backwards with regard to traditional mathematical convention, we receive the dyadic expansion of the first 15 integers and beginning with zero listed upside down. Note that this yields an obvious representation of all subsets of a set of 4 elements or rather the incidence vectors of these sets.

More generally, the mapping

$$\tau : \mathbb{N}_0^n \times \mathcal{A}^K \rightarrow \mathcal{A}^{K+1}$$

is given by

$$(b_n, \ldots, b_0)\tau A = B$$

with

$$B : [1, \ldots, n+1] \times I_1 \times \ldots I_K \rightarrow \mathbb{R}$$
and

(11) \[ B[i_1, \ldots, i_K] := (b_n, \ldots, b_0)TA/i_1, \ldots, i_K)(I_l \in I(l = 1, \ldots, K) \]
8 Grade Up and Grade Down

As previously we denote by \( A^1 \) the space of vectors and by \( N^1 \) the space of integer vectors. The function \( \phi \) is called \textit{grade up} and is defined as follows:

**Definition 8.1.**

(1) \( \phi : A^1 \rightarrow N^1 \).

For \( x = (x_1, \ldots, x_n) \) satisfying

\[
x_{i_1} < x_{i_2} < \ldots < x_{i_n}
\]

with \( \{i_1, \ldots, i_n\} = \{1, \ldots, n\} \), we define

\[
\phi x := (i_1, \ldots, i_n).
\]

If some coordinates are equal, the function respects the original ordering indicated by \( x \).

That is, \( \phi \) orders the indices \( 1, \ldots, n \) according to the size of the coordinates of \( x \). As a consequence,

\[
x[\phi x] = (x_{i_1}, \ldots, x_{i_n})
\]

arranges the coordinates of \( x \) in increasing order and

\[
x[(\phi x)[1]]
\]

is the minimum of the coordinates of \( x \).

**Example 8.2.**

\[
\begin{align*}
\text{X} &\leftarrow 3.1 \ (20\div3)\ 1.1\ 0 \\
 3.1 &\ 6.666666667\ 1.1\ 0 \\
\phi X &
\end{align*}
\]

\[
\begin{align*}
4 &\ 3\ 1\ 2 \\
X[\phi X] &
\end{align*}
\]

\[
\begin{align*}
0 &\ 1.1\ 3.1\ 6.666666667 \\
X[(\phi X)[1]] &
\end{align*}
\]

0
Let us discuss the action of $\Phi$ in connection with permutations. We would like to represent tentatively a permutation by a vector. That is, if the permutation $\pi$ is more elaborately given by

$$\pi = \begin{pmatrix} 1 & 2 & \ldots & i & \ldots & n \\ \pi(1) & \pi(2) & \ldots & \pi(i) & \ldots & \pi(n) \end{pmatrix},$$

then we represent this permutation by the vector in the lower row, i.e., by $(\pi(1), \pi(2), \ldots, \pi(i), \ldots, \pi(n))$.

Now, if $\pi(i) = k$ holds true, then coordinate $i$ of the vector $\pi$ is the $k^{th}$ in order, i.e.,

$$(\Phi)[k] = i.$$
However, the permutation $\pi^{-1}$ as described by

\[
\pi^{-1} = \left( \begin{array}{cccc}
1 & 2 & \ldots & i \\
\pi^{-1}(1) & \pi^{-1}(2) & \ldots & \pi^{-1}(i) \\
\vdots & \vdots & \ddots & \vdots \\
\pi^{-1}(1) & \pi^{-1}(2) & \ldots & \pi^{-1}(n) \\
\end{array} \right),
\]

also satisfies $\pi^{-1}(k) = i$. From this we conclude that

$$\Phi\pi = \pi^{-1}$$

provided we interpret both sides as permutations.

A permutation $\pi$ acts as well as a linear mapping $\pi : \mathbb{R}^n \to \mathbb{R}^n$, the appropriate definition is

\[
(\pi(x))_i := x_{\pi^{-1}(i)} \quad (x \in \mathbb{R}^n, \ i \in I).
\]

This allows to interpret $\pi$ as a “permutation of the axes” of $\mathbb{R}^n$. Now, if $X \in \mathcal{A}^1$ represents $x \in \mathbb{R}^n$ in $\mathcal{A}_P\mathcal{L}$ -code, then, as $\pi^{-1}$ corresponds to $\Phi\pi$, we obtain

\[
X[\Phi\pi]_i = X[\Phi\pi[i]] = X[\pi^{-1}(i)] = x_{\pi^{-1}(i)}.
\]

Hence, the linear mapping $\pi : \mathbb{R}^n \to \mathbb{R}^n$, is given by $\pi(x) = X[\Phi\pi]$. 
Section 9: The Dyadic Transposition \( \varphi \)

A permutation \( \pi : \{1, \ldots, K\} \to \{1, \ldots, K\} \) may be described by

\[
\pi = \begin{pmatrix}
1 & 2 & \cdots & i & \cdots & n \\
\pi(1) & \pi(2) & \cdots & \pi(i) & \cdots & \pi(n)
\end{pmatrix},
\]

and is canonically represented by the vector in the lower row, i.e., by

\((\pi(1), \pi(2), \ldots, \pi(i), \ldots, \pi(n))\).

We shall now describe the action of permutations on arrays which result from the corresponding permutations of the intervals which constitute the domain of definition.

**Definition 9.1.** Let \( \pi \) be a permutation of \( \{1, \ldots, K\} \) and let \( A \in \mathbb{A}^K \), say

\[
A : I_1 \times \cdots \times I_K \to \mathbb{R},
\]

then the array

\[
\pi \varphi A : I_{\pi^{-1}(1)} \times \cdots \times I_{\pi^{-1}(K)} \to \mathbb{R}
\]

is defined by

\[
\pi \varphi A[j_1; \ldots; j_K] := A[j_{\pi(1)}; \ldots; j_{\pi(K)}].
\]

The reader should make sure that the operation is well defined. In particular, one has to check that the arguments of the transposed operator correspond to the correct intervals. Indeed, if

\[
j_l \in I_{\pi^{-1}(l)}
\]

then

\[
j_{\pi(l)} \in I_{\pi(\pi^{-1}(l))} = I_l,
\]

thus, \( \pi \varphi A \) is well defined.

Now we observe that the permutation \( \pi \) acts canonically on the dimension vector of the array while transposition is applied. More detailed, the vector \( \rho A \) is canonically permuted in order to yield the dimension vector of the transposed array. Indeed, we have

\[
(\rho \pi \varphi A)_l = |I_{\pi^{-1}(l)}| = (\rho A)_{\pi^{-1}(l)} = (\pi \rho A)_l.
\]
That is the vector \( \mathbf{x} = \rho \mathbf{A} \) is nicely permuted via the usual convention \(((\pi(\mathbf{x}))_i = x_{\pi^{-1}(i)})\) in order to obtain the dimension vector \( \rho \) of the transposed version of \( \mathbf{A} \):

\[
\rho \pi \mathbf{A} = \pi \rho \mathbf{A}.
\]

Now, as we have described the action of permutations by means of the operator \textit{grade up} \( \Phi \) in \textbf{Section 8}, we expect that \( \pi(\mathbf{x}) \) is represented in \textsc{APL} code by \( \mathbf{x}[\Phi \pi] \). Thus, we expect the equation

\[
\rho \pi \mathbf{A} = (\rho \mathbf{A})[\Phi \pi].
\]

\textbf{Example 9.2.} We take a very simple array which however demonstrates the effects nicely:

\[
\begin{array}{ccc}
A+1 & 2 & 3 \\
10 & 20 & 30 \\
40 & 50 & 60 \\
\end{array}
\]

\[
\begin{array}{c}
\text{PI} \\
2 & 3 & 1 \\
\end{array}
\]

\[
\begin{array}{c}
\rho \text{ PI } \Phi \mathbf{A} \\
10 & 40 \\
20 & 50 \\
30 & 60 \\
\end{array}
\]

\[
\begin{array}{c}
\rho \text{ PI } \Phi \mathbf{A} \\
3 & 1 & 2 \\
\end{array}
\]

\[
(\rho \mathbf{A})[\Phi \text{PI}] \\
3 & 1 & 2 \\
\]

It should be stressed that the application of \( \pi \) on vectors is suitably the one that acts inversely on the coordinates (the above mentioned convention).
For one argument, this action exactly reflects the permutation of axis as indicated by \( \pi \): e.g., if the axes of \( \mathbb{R}^3 \) are cyclically permuted by \( \pi = (2, 3, 1) \) then the vector \( \mathbf{x} = (1, 2, 3) \) is thrown into \( \pi \mathbf{x} = (3, 2, 1) \). For a second argument, if \( \mathbf{v} \) is a set function (a game) defined on the subsets of \( I \), then the appropriate action induced by \( \pi \) yields the game \( \pi \mathbf{v} \) which is given by \( \pi \mathbf{v}(S) := \mathbf{v}(\pi^{-1}(S)) \) (\( S \subseteq I \)). For set functions, this convention is preferable as \( \pi^{-1} \) preserves all set-operations and hence preserves additivity of a set function once it occurs. The action on vectors and the one on set functions correspond nicely in this case if a vector is seen as an additive set function.

The dyadic transposition may be generalized to the case that \( \pi \) is not a permutation but just a surjective mapping. Let \( L \leq K \) and let

\[ \pi : \{1, \ldots, K\} \to \{1, \ldots, L\} \]

be a surjective mapping. Define

\[ I_l^\pi := \bigcap_{k \in \pi^{-1}\{l\}} I_k = [1, \min_{k \in \pi^{-1}\{l\}} \rho_k] \]

(such that \( I_l^\pi = I_{\pi^{-1}\{l\}} \) holds true whenever \( \pi \) is bijective), then we define

\[ \pi \mathbf{A} : I_1^\pi \times \ldots \times I_L^\pi \to \mathbb{R} \]

\[ \pi \mathbf{A}[j_1; \ldots; j_L] := \mathbf{A}[j_{\pi(1)}; \ldots; j_{\pi(L)}]. \]

Observe the reaction of the dimension vector, the relation

\[ \rho \pi \mathbf{A} = \pi (\rho \mathbf{A}). \]

generalizes the bijective case suitably, if we use the convention

\[ \pi(\mathbf{x})_l := \min_{k \in \pi^{-1}\{l\}} \rho_k \]

in order to characterize the vector \( \pi \mathbf{x} \). This is verified via

\[ (\rho \pi \mathbf{A})_l = \min\{\rho_k | \pi(k) = l\} = \min_{k \in \pi^{-1}\{l\}} \rho_k. \]

In particular consider the mapping \( \pi : \{1, \ldots, K\} \to \{1\} \) which is suitably represented by the vector \( (1, \ldots, 1) \) or \( K \cdot 1 \), then

\[ \pi \mathbf{A}[j_1] = \mathbf{A}[j_{\pi(1)}; \ldots; j_{\pi(K)}] = \mathbf{A}[j_1; \ldots; j_1] \]

or for short

\[ 1 1 \ldots 1 \mathbf{A}[i] = (K \cdot 1) \mathbf{A}[i] = \mathbf{A}[i; \ldots; i] \]
meaning that the application yields the diagonal, if \( K = 2 \) and hence \( A \) is a matrix, then \( 1 1 1 \varphi A \) is the vector of diagonal elements.

Similarly, if \( \pi\{1, 2, 3\} \to \{1, 2\} \) is given by (122), then

\[
1 2 2 \varphi A[j_1, j_2] = A[j_1, j_2, j_2]
\]
or for short

\[
1 2 2 \varphi A[i, j] = A[i, j, j].
\]

Analogously

\[
2 1 1 \varphi A[i, j] = A[j, i, i].
\]

Example 9.3.

\[
A
\]

\[
\begin{array}{cccccc}
111 & 112 & 113 & 114 & 115 \\
121 & 122 & 123 & 124 & 125 \\
131 & 132 & 133 & 134 & 135 \\
\end{array}
\]

\[
211 \ 212 \ 213 \ 214 \ 215 \\
221 \ 222 \ 223 \ 224 \ 225 \\
231 \ 232 \ 233 \ 234 \ 235 \\
\]

\[
1 \ 3 \ 2 \varphi A
\]

\[
\begin{array}{ccc}
111 & 121 & 131 \\
112 & 122 & 132 \\
113 & 123 & 133 \\
114 & 124 & 134 \\
115 & 125 & 135 \\
\end{array}
\]

\[
211 \ 221 \ 231 \\
212 \ 222 \ 232 \\
213 \ 223 \ 233 \\
214 \ 224 \ 234 \\
215 \ 225 \ 235 \\
\]

\[
1 \ 2 \ 2 \varphi A
\]

\[
\begin{array}{ccc}
111 & 122 & 133 \\
211 & 222 & 233 \\
\end{array}
\]

.... takes the diagonal of the first
and second partial matrix!

and second partial matrix:

\[
A = \begin{pmatrix}
111 & 112 & 113 & 114 & 115 \\
121 & 122 & 123 & 124 & 125 \\
131 & 132 & 133 & 134 & 135 \\
211 & 212 & 213 & 214 & 215 \\
221 & 222 & 223 & 224 & 225 \\
231 & 232 & 233 & 234 & 235 \\
\end{pmatrix}
\]
Chapter 4

Operators

Operators are maps defined on certain function spaces. This rather vague definition can be made precise for a specific operator by indication the domain of definition and the range of the mapping as usual. However, as these quantities are not really present in the APL workspace, it is now frequently not too rewarding to actually present a list of the details. Thus, one frequently explains the action of an operator by indicating the arguments (functions) and the resulting images (also functions) via some direct presentation – not including the precise domain of definition etc.
1 Ravel

We start with a rather simple example of an “operator”, called *ravel* and represented by the slash */*. A preliminary definition would be given as follows.

**Definition 1.1.** Let \( f : \mathbb{R}^2 \rightarrow \mathbb{R} \) be a function. Then, for every \( n \in \mathbb{N} \) the function

\[
 f/* : \mathbb{R}^n \rightarrow \mathbb{R}
\]

is defined by means of the following recursion:

For \( n = 2 \) the function \( f/* \) equals \( f \).

For \( n > 2 \) the function \( f/* : \mathbb{R}^n \rightarrow \mathbb{R} \) is given via

\[
 f/*(x_1, \ldots, x_n) := f(x_1, f/*(x_2, \ldots, x_n)).
\]

Note that the operation is not commutative, in general the term

\( f(f(x_1, \ldots, x_{n-1}), x_n) \)

may well differ from the above.

Now, it would be more precise to introduce, say, the set \( \mathcal{F}^n := \{ f| f : \mathbb{R}^n \rightarrow \mathbb{R} \} \) and to argue that */* or rather • 7 is a function

\[
 \bullet*/ : \mathcal{F}^2 \rightarrow \bigcup_{n=1}^{\infty} \mathcal{F}^n.
\]

However, the gain in precision is not very high compared to the loss in overview and besides, in order to define the class \( \mathcal{F}^n \) one would have to say a few words about “functions” or “all functions available in *APL*” which then leads possibly to more confusion. We shall, therefore, continue discussing operators more informally and more to the liking of the computer scientist.

**Example 1.2.** The most simple functions on \( \mathbb{R}^2 \) are the dyadic operations +, − etc. Clearly, for \( x \in \mathbb{R}^n \) we have

\[
 +(x_1, \ldots, x_n) = x_1 + \ldots + x_n
\]

and

\[
 -(x_1, \ldots, x_n) = x_1 - (\ldots - (x_{n-2} - (x_{n-1} - x_n))).
\]
This works with all further dyadic functions like $\star$, $\land$, $\lor$ ....

As a first generalization we consider an operator $\bullet /_{[L]}$ which is also acting on functions. To this end let $A \in A^K$ be an array, i.e.,

$$A : I_1 \times \ldots \times I_K \to \mathbb{R}$$

and let $L$ be such that $1 \leq L \leq K$. Now, for fixed choice of indices $(i_1, \ldots, i_{(L-1)}, i_{(L+1)}, \ldots, i_K)$ we obtain the vector

$$A(i_1, \ldots, i_{(L-1)}, \bullet, i_{(L+1)}, \ldots, i_K) \in \mathbb{R}^{I_L},$$

hence, whenever $f$ is a function, then the function $f /$ admits this vector as an argument, hence the term

$$C(i_1, \ldots, i_{(L-1)}, i_{(L+1)}, \ldots, i_K) := f(A(i_1, \ldots, i_{(L-1)}, \bullet, i_{(L+1)}, \ldots, i_K))$$

is well defined. Obviously, $C$ is an array, $C \in A^{K-1}$. This now defines the value of the function $f /_{[L]}$ applied on the array $A$, i.e.,

$$C(i_1, \ldots, i_{(L-1)}, i_{(L+1)}, \ldots, i_K) =: (f /_{[L]}(A))(i_1, \ldots, i_{(L-1)}, i_{(L+1)}, \ldots, i_K)$$

or rather

$$C := f /_{[L]}(A).$$

Combining, we have defined a function

$$f /_{[L]}(A) : A^K \to A^{K-1},$$

and it is left to the reader to ponder about the operator $/_{[L]}$, it’s domain and it’s range .... We see that the operator ‘ravels’ along the coordinate $L$ and keeps all other coordinates unchanged.

**Remark 1.3.** Two special cases are noteworthy, we have

$$f /_{[1]} = f / \text{ and } f /_{[K]} = f f,$$

that is, $f /$ may be seen to work on arrays as well (not just on vectors as in the initial definition), ravelling along the first coordinate. Similarly $f f$ works on arrays, ravelling the last coordinate (!).

Consider the following obvious results.

**Example 1.4.**
10
1 2 3 4 5 6 7 8 9 10
+/- 1 2 3 4 5 6 7 8 9 10
55

(10 \times (10+1)) \div 2
55

and

-/- 1 2 3 4 5 6 7 8 9 10
-5
-/- 1 2 3 4 5 6 7 8 9 10 11
6

And furthermore

A
6 7 8 1 10
11 12 13 1 15
0 0 0 1 1
21 22 23 1 25

+/[1]A
38 41 44 3 50
+/[2]A
32 52 0 92
2 Outer Product

The outer product denoted by $\odot$, can be seen as a version of the tensor product used in various contexts with physical background or in tensor analysis. $\odot$ is as well a map on functions of two variables (i.e., dyadic functions in APL slang). The result is a function of two variables, mainly, of two arrays. We omit a formal definition of the range. Note that the function which serves as an argument appears on the right side of the operator, so we are tempted to use the notation $\odot \bullet$, which should give raise to some obvious questions in view of the order of execution in APL.

Consider a function $f \in \mathcal{F}^2$. Then, for any $K, L \in \mathbb{N}$ the function

$$\odot f$$

is defined on pairs of arrays, say $A \in \mathcal{A}^K, B \in \mathcal{A}^L$, and yields another array, say

$$A (\odot f) B =: C .$$

Now, if $A : I_1 \times \ldots \times I_K \to \mathbb{R}$ and $B : J_1 \times \ldots \times J_L \to \mathbb{R}$ is the case, then $C \in \mathcal{A}^{K+L}$ is an array of the form

$$C : I_1 \times \ldots \times I_K \times J_1 \times \ldots \times J_L \to \mathbb{R}$$

which is given via

$$C(i_1, \ldots, i_K, j_1, \ldots, j_L) := A(i_1, \ldots, i_K) f B(j_1, \ldots, j_L)$$

for all

$$(i_1, \ldots, i_K, j_1, \ldots, j_L) \in I_1 \times \ldots \times I_K \times J_1 \times \ldots \times J_L .$$

In particular, if the arrays involved are vectors, say $x$ and $y$, then we find that $C := x (\odot f) y$ is a matrix given by

$$C_{i,j} = C(i, j) = x_i f y_j = f(x_i, y_j) .$$

Introducing the particular version hen $f = \times$ holds true, we see that

$$C_{i,j} = x_i \times y_j = x_i y_j$$

resembles indeed the most familiar version of a tensor product available.

**Example 2.1.** The outer product of two vectors may as well be defined for the function $+$:
10 \times 15
10 20 30 40 50
18
1 2 3 4 5 6 7 8

(10 \times 15) \odot + 18
11 12 13 14 15 16 17 18
21 22 23 24 25 26 27 28
31 32 33 34 35 36 37 38
41 42 43 44 45 46 47 48
51 52 53 54 55 56 57 58

and slightly generalized

(100 \times 13) \odot + (10 \times 15) \odot + 18
111 112 113 114 115 116 117 118
121 122 123 124 125 126 127 128
131 132 133 134 135 136 137 138
141 142 143 144 145 146 147 148
151 152 153 154 155 156 157 158

211 212 213 214 215 216 217 218
221 222 223 224 225 226 227 228
231 232 233 234 235 236 237 238
241 242 243 244 245 246 247 248
251 252 253 254 255 256 257 258

311 312 313 314 315 316 317 318
321 322 323 324 325 326 327 328
331 332 333 334 335 336 337 338
341 342 343 344 345 346 347 348
351 352 353 354 355 356 357 358

7 1 3 4 0 11 1 \odot \leq 18
0 0 0 0 0 0 1 1
also, if both vectors are equal:

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\]

\[\wedge \]

\[
(\tau_6)^{\circ} \leq \tau_6
\]

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

\[
(\tau_6)^{\circ}.< \tau_6
\]

\[
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

and we obtain the identity matrix via

\[
(\tau_6)^{\circ} = \tau_6
\]

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\]
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
3 Inner Product

The inner product is a generalization of the concept as defined for vectors. It works on functions and the result is an inner (scalar) product on arrays based on these two functions. As previously we start out with (in this case two) functions defined for two variables, “dyadic” in APL slang. The result of the mapping “inner product” is one dyadic function.

The notation used is a dot ’ . ’ or rather ’ ⋅ ⋅ ’ as it is employed frequently for the inner (scalar) product of vectors. And indeed, it is convenient to implement a recursive definition that starts out with an operation on pairs of vectors, i.e., the traditional scalar product. The operator / (Ravel) is involved in the following definition.

**Definition 3.1.** Let $f, g \in \mathcal{F}^2$. Then the function $f.g \in \mathcal{F}$ is defined by

$$x(f.g)y = f(x_1y_1, \ldots, x_ny_n)$$

for any pairs of vectors $x = (x_1 \ldots x_n)$ and $y = (y_1 \ldots y_n)$ of equal length.

Note that for $f = +$ our definition reduces to

$$x(+.g)y = +/(x_1y_1, \ldots, x_ny_n) = x_1y_1 + \ldots + x_ny_n,$$

and hence when $g$ happens to be the product, i.e., $g = \times$, then we find

$$x(+.\times)y = +/(x_1\times y_1, \ldots, x_n\times y_n) = x_1 \times y_1 + \ldots + x_n \times y_n,$$

which is indeed the traditional inner or scalar product on vectors in $\mathbb{R}^n$.

Now we continue with the definition of the inner product as follows.

**Definition 3.2.** Let $f \in \mathcal{F}, g \in \mathcal{F}$ be a pair of (dyadic) functions. Then, the function $f.g \in \mathcal{F}$ is given as follows: for any pair of arrays $A : I_1 \times \ldots \times I_K \rightarrow \mathbb{R}$ and $B : J_1 \times \ldots \times J_L \rightarrow \mathbb{R}$ such that

$$I_K = J_1$$

holds true, the array $C = A(f.g)B$ is described by

$$C : I_1 \times \ldots \times I_{(K-1)} \times J_2 \times \ldots \times J_L \rightarrow \mathbb{R}$$

$$C(i_1 \ldots i_{K-1}, j_2 \ldots j_L) = A(i_1 \ldots i_{K-1}, \bullet) (f.g)B(\bullet, j_2 \ldots j_L)$$

for all

$$(i_1 \ldots i_{K-1}, j_2 \ldots j_L) \in I_1 \times \ldots \times I_{(K-1)} \times J_2 \times \ldots \times J_L;$$
here the second line of (2) of course refers to the definition provided in (1).

We specify this general definition immediately to the case in which \( f = + \) and \( g = \times \). Assume also that \( A \) and \( B \) are matrices. Then we find for \((i, j) \in I_1 \times J_2:\)

\[
(A + . \times B)(i, j) = A_{i \bullet} + . \times B_{\bullet j}
\]

\[
+/A_{i \bullet} \times B_{\bullet j} = (AB)_{ij},
\]

which shows that \( A + . \times B = AB \) is the conventional matrix product.

We demonstrate this by means of the following operations:

\[
(10 \times 15) \circ . + 18
\]

\[
\begin{array}{cccccccc}
11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\
21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \\
31 & 32 & 33 & 34 & 35 & 36 & 37 & 38 \\
41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\
51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 \\
\end{array}
\]

\[
(100 \times 13) \circ . + (10 \times 15) \circ . + 18
\]

\[
\begin{array}{cccccccccccccccc}
111 & 112 & 113 & 114 & 115 & 116 & 117 & 118 \\
121 & 122 & 123 & 124 & 125 & 126 & 127 & 128 \\
131 & 132 & 133 & 134 & 135 & 136 & 137 & 138 \\
141 & 142 & 143 & 144 & 145 & 146 & 147 & 148 \\
151 & 152 & 153 & 154 & 155 & 156 & 157 & 158 \\
\end{array}
\]

\[
211 & 212 & 213 & 214 & 215 & 216 & 217 & 218 \\
221 & 222 & 223 & 224 & 225 & 226 & 227 & 228 \\
231 & 232 & 233 & 234 & 235 & 236 & 237 & 238 \\
241 & 242 & 243 & 244 & 245 & 246 & 247 & 248 \\
251 & 252 & 253 & 254 & 255 & 256 & 257 & 258 \\
\end{array}
\]

\[
311 & 312 & 313 & 314 & 315 & 316 & 317 & 318 \\
321 & 322 & 323 & 324 & 325 & 326 & 327 & 328 \\
331 & 332 & 333 & 334 & 335 & 336 & 337 & 338 \\
341 & 342 & 343 & 344 & 345 & 346 & 347 & 348 \\
\end{array}
\]
Also, the inner product can be used to generate a unit matrix or its relatives:

\[(\mathbf{110})^\circ \cdot = (\mathbf{110})\]
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

\[(\mathbf{110})^\circ \cdot \leq (\mathbf{110})\]
\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]
Chapter 5

Programming

Now we would like to generate additional functions we are interested in. This technique is what in everyday language is called “Programming”. Technically the result of writing a “program” with respect to APL syntax is a function in the sense of Chapter 3. That is, functions defined by programs work on arguments that may be vectors or arrays.
1 The Definition of Functions – Programming in APL

The definition of a function takes place within “definition mode”. The shape of the function is indicated by the interior of a neighborhood delimited by

0 Function header
1 ...
2 ...
...
[..] ...
...
[n] ...

However, in newer versions there are editors available in order to put the system into “definition mode.”

There are

1. Functions one, two, and no variable.
2. Functions with or without assignment
3. Functions with global and local variables

Functions are identified by a name. Within the definition, the name and the above characteristica with respect to variables and assignments have to be specified.

Functions are referred to by name and, if applicable, by indicating the variables they depend on. This amounts specifying arguments as we do for functions like \( \tau \), \( \rho \), or \( \pi \).

Example 1.1. We start with a standard problem, the solutions of a quadratic equation \( x^2 + ax + b = 0 \) is given by

\[
x_{1,2} = \frac{-a \pm \sqrt{a^2 - 4b}}{2},
\]

provided the discriminant (the term under the root) is nonnegative.

As a first attempt we write a function/program for two variables \( a \) and \( b \) that yields the solution. This amounts to a function of two variables \( a \) and
$b$, which in the solution of 1 as its output. This output can be assigned to a variable.

\[
\begin{align*}
[0] & \ R+ \ A \ \mathrm{QUEQ} \ B \\
[1] & \ D+(A*2) - 4xB \\
[2] & \ R+((-A) + (1,-1) \times( D \times 0.5))\div2
\end{align*}
\]

Clearly, the function will come up with an ERROR code if the discriminant happens to be negative. Thus, we cause the interpreter to exit the program (i.e., to move to line [0]) if this problem occurs:

\[
\begin{align*}
[0] & \ R+ \ A \ \mathrm{QUEQ} \ B \\
[1] & \ D+(A*2) - 4xB \\
[2] & \ \to(D < 0)/0 \\
[3] & \ R+((-A) + (1,-1) \times( D \times 0.5))\div2
\end{align*}
\]

Now, this function ends up without any comment and result whenever the discriminant is negative. This can be mended by sending the compiler to a line with label 'END' whenever that happens.

\[
\begin{align*}
[0] & \ R \ + \ A \ \mathrm{QUEQ} \ B \\
[1] & \ D \ + \ (A*2) - 4xB \\
[2] & \ \to(D < 0)/END \\
[3] & \ R \ + \ ((-A) + (1,-1) \times( D \times 0.5))\div2 \\
[4] & \ \to0 \\
[5] & \ \mathrm{END:} \\
[6] & \ '\mathrm{DISCRIMINANT\ NEGATIVE}' \\
[7] & \ '0\ \mathrm{ASSIGNED\ TO\ SOLUTION\ OF\ QUEQ}' \\
[8] & \ R \ + \ 0
\end{align*}
\]

and if we execute this we obtain
Example 1.2. There may be a good reason to avoid an ERROR message without explicitly testing for the value of a certain variable. E.g. the monadic APL function yields the inverse of a matrix, provided the matrix is nonsingular, i.e., the determinant does not vanish. Now, when applying this function we don’t want to compute the determinant. On one hand, right now we do not even have a program available which performs this task. And on the other hand, even if we had a program, we would prefer not to waste time with computing a determinant. Because, whatever the function does, it is probably much faster that any self constructed program we can possible arrange for at this stage. The following method indicates how to avoid the ERROR message:

```
7 APLAG SAVED 04/10/2002 17:10:31

\ERRCATCH[]\V
[0] INV + ERRCATCH A;ELX
[1] \ This function demonstrates the holdoff of ERRORS.
[2] \ Without action the function DM is assigned to
[3] \ DM -- and is executed whenever
[4] \ an ERROR occurs. In this case DM replies by
[5] \ announcing the type of ERROR.
[6] \ Within the present program
[7] \ '->ERR' is assigned to ELX --.
[8]
[9] \ELX+ '->ERR'
[10] INV+\A
[11] \0
[12] ERR:
```
Section 1: The Definition of Functions -- Programming in APL

[13] 'NO INVERSE EXISTS'
[14] INV+ (pA) ρ0
   ERRCATCH 2 3 ρ 110
NO INVERSE EXISTS
0 0 0
0 0 0
   MAU+ 2 2 ρ 1 1 1
   ERRCATCH MAU
NO INVERSE EXISTS
0 0
0 0
   M
1 1
-1 1
   ERRCATCH M
0.5 -0.5
0.5 0.5
   ERRCATCH 3 2 ρ 110
-1.333333333 0.333333333 0.666666667
1.083333333 0.333333333 -0.416666667

The above example shows a function of two variables with assignment. Generally the following forms appear:

<table>
<thead>
<tr>
<th>without assignment:</th>
<th>NILADIC</th>
<th>MONADIC</th>
<th>DYADIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>with assignment:</td>
<td>FN</td>
<td>FN Y</td>
<td>X FN Y</td>
</tr>
<tr>
<td></td>
<td>R +FN</td>
<td>R + FN Y</td>
<td>R +X FN Y</td>
</tr>
</tbody>
</table>

The variables X, Y, R are local: they appear in the programming and execution mode but not in WORKSPACE. Further Variables can be made local by explicitely registering them in the header, separated by a semicolon.

Branching can be set up in a program by refering the interpreter to move to separate labels depending on arguments.

Example 1.3. The Newton method asks for the computation of \( \sqrt{2} \) via

\[
a_0 = 2; \quad a_{n+1} = \frac{1}{2} a_n + \frac{1}{a_n} \quad (n = 1, 2, \ldots)
\]

\[
\sqrt{2} = \lim_{n \to \infty} a_n
\]
This is reflected as follows:

```
[ 0] A + NEWTON L; N
[ 1] A= 2,(L-1)p0
[ 2] N = 2
[ 3] "BUILDING A LOOP:
[ 4] REP:
[ 6] "BRANCHING:
[ 7] → (REP,END)[1+L ≤ N + N+1]
[ 8] END:
[ 9] 'The sequence is:'
[10] A
```
Chapter 6

The Coalitional Function

We are going to introduce central topics of Cooperative Game Theory in the context of APL programming.
1 Games

**Coalitional function** Cooperative Game Theory deals with nonadditive set functions interpreted as coalitional functions or just games. More precisely, a game is a triple

\[ \Delta := (I, P, v) \]

with the following ingredients: \( I := \{1, \ldots, n\} \) is a finite set, the set of **players**. \( P := \{S \mid S \subseteq I\} \) is the power set of \( I \) and interpreted as the system of (feasible) **coalitions**. Finally,

\[ v : P \rightarrow \mathbb{R}, \quad v(\emptyset) = 0 \]

is the **coalitional function**. This function attaches to every coalition (group of players) \( S \) a real ("monetary") value \( v(S) \), the "worth" of \( S \). Coalition \( S \) can achieve \( v(S) \) by cooperation within some economic, political, social, ... context, the details are not specified. Thus, the set function \( v \) provides the "incentive" for players to form coalitions. By a standard convention, the empty coalition achieves worth 0. A special case is provided by a **simple game** which is given whenever \( v \) takes only values 0 and 1, thus

\[ v : P \rightarrow \{0, 1\}, \quad v(\emptyset) = 0. \]

We attempt to discuss the appropriate APL –version of a game. First of all, we identify coalitions \( S \in P \) with the incidence vector or **indicator** \( 1_S \) which is given by

\[ 1_S(i) = 1 \quad (i \in I), \quad 1_S(i) = 0 \quad (i \notin I). \]

we refer to this vector also as to the **profile** of \( S \). Clearly, this concept is immediately transferred to the APL –context; a coalition or rather a profile of a coalition is an \( n \)–vector of zeros and ones. The power set or system of coalitions \( P \) is hence given by a matrix of all possible \( 0 – 1 \)–vectors. This is obtained by decoding the numbers \( 0, 1, \ldots, 2^n – 1 \) with respect to the dyadic basis. We call this the "profile matrix". The program "PROFILES" achieves this as follows:

```
\texttt{PROFILES[[]]}
```
### Section 1: Games

[0] Z+PROFILES N

Generates the profiles of all coalitions of N players.

[2] The result is a 0-1-matrix; N rows, 2*N columns.

Standard, elegant - but exhausts the Workspace at 21 players.


[5] ************************************************************

[6]

[7] Z+(Np2)\textsuperscript{\tau}1+\textsuperscript{\tau}2+N

PROFILES 5

\begin{verbatim}
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
\end{verbatim}

The coalitions appear as the columns of the matrix PROFILES ordered according to the ordering of the natural numbers which are dydically represented. As a result, the lexicographic ordering is reversed, i.e., player $N$ appears first and the grand coalition last. (The lexicographic ordering ranks 0 above 1).

We list two alternatives of generating PROFILES which may seem less elegant from the APL –puristic viewpoint but are faster and more efficient. The reader is obliged to maske himself acquainted with the inherent structure.

∀PROFAST[[]]∀

[0] P+PROFAST N;I;L

Generates profiles fast.

LIMIT ERROR at $N = 24$

[4] ************************************************************

[5] P+(0,2*N)p0

[6] I+N
Now to the coalitional function. In APL context the coalitional function

\[ v : \mathbb{P} \to \mathbb{R} \]
is now represented by a function
\[ c : \{1_S | S \in \mathbb{P}\} = \{0, 1\}^N \rightarrow \mathbb{R} \]
or rather an array
\[ C : I_1 \times \ldots \times I_N \rightarrow \mathbb{R} \]
with
\[ I_1 = \ldots = I_N = [0, 1] . \]
As far as the index origin has been set to be \( \square \mathbf{I} 0 \cdot 0 \), this convention is sufficient to describe games.

However, usually we prefer to have \( \square \mathbf{I} 0 \cdot 1 \), in which case we should use an array
\[ C : I_1 \times \ldots \times I_N \rightarrow \mathbb{R} \]
with
\[ I_1 = \ldots = I_N = [1, 2] . \]
Both versions are in a sense isomorphic as mathematical objects so that we can indeed use any array on \( I_1 \times \ldots \times I_N = [1, 2]^N \) i.e., any \( C \) with \( \rho C = (2, \ldots, 2) = N \rho 2 \) for representation of a “game”. Note however that, in the second case the arguments have to be changed. In the first version the argument if \( C \) is just the 0–1-vector \( 1_S \). In the second, one has to choose \( 1_S + 1_I \) so as to obtain a 1–2-vector representing coalition \( S \). Therefore, in \( \mathsf{APL} \) code, \( v(S) \) writes \( c[1_S + 1_I] \).

There is a second approach to the coalitional function as an object in the Workspace, simpler and for most purposes sufficient. To describe this approach, observer that the function \( v : \mathbb{P} \rightarrow \mathbb{R} \) is as well described by a vector \( (v(S))_{S \in \mathbb{P}} \) which is of length \( 2^n \). Hence, a coalitional function in \( \mathsf{APL} \) code could be just an vector \( V \) with dimension \( \rho V = 2 \times N \) (= \( 2^n \)). In this case, the ordering of the coalition has to be fixed once for all so that the coordinates of \( V \) seen as a vector are in unique correspondence to the elements of \( \mathbb{P} \). A most natural way would be to use the orderung indicated by the matrix PROFILES.

**Example 1.1.** Consider the following two programs written to generate additive games.
\[ V \leftarrow \text{ADDI } W \]

\[ \text{\(\star\)} \]

Elegant version. Generates the ADDITIVE coalition function corresponding to the measure indicated by the vector \( W \) (the weights).

\[ \text{\(\star\)} \]

Generates a \( 2 \times N \) vector representing the coalition function in the order of coalitions prescribed by the matrix PROFILES.

\[ \text{\(\star\)} \]

\[ \text{\(\star\)} \]

\[ V \leftarrow W + \times \text{PROFILES } \rho W \]

```
ADDI 1 4 7 12
0 12 7 19 4 16 11 23 1 13 8 20 5 17 12 24
```

This attached to the matrix PROFILES 4:

```
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 1 1
```

\[ \text{\(\star\)} \]

\[ \text{\(\star\)} \]

\[ \text{\(\star\)} \]

\[ V \leftarrow \text{ADDITIV}[\text{\(\star\)}] V \]

\[ \text{\(\star\)} \]

\[ V \leftarrow \text{ADDITIV } W;N;L;\text{PROFZEIL};\text{Q} \]

\[ \text{\(\star\)} \]

Generates the coalitional function resulting from a measure/Vector of weights \( W \).

\[ \text{\(\star\)} \]

Input: Vector of weights \( W = (W[1], \ldots, W[N]) \).

\[ \text{\(\star\)} \]

Output: Funktion/ Game \( V \) (additive) - as a \( 2 \times N \) vector \( V[0] \) .... \( V[2+N] \) - in the ordering according to PROFILES.

\[ \text{\(\star\)} \]

\[ \text{\(\star\)} \]

\[ \text{\(\star\)} \]

\[ L \leftarrow 2 \times N \times \rho W \]

\[ \text{\(\star\)} \]

\[ V \leftarrow L \rho 0 \]

\[ \text{\(\star\)} \]

\[ I \leftarrow 1 \]
[12] \text{UP} \rightarrow (N=I+I+1)/0
[13] Q \leftarrow 2 \times I
[14] \text{PROFCLMN} \leftarrow \rho(Qp0), Qp1
[15] V+V \rightarrow \text{PROFCLMN} \times W[N-I]
[16] \rightarrow \text{UP}

\text{ADDITIV 1 4 7 12}
0 12 7 19 4 16 11 23 1 13 8 20 5 17 12 24
Chapter 7

Cephoids

Within this chapter we provide a library for the computation of the data of a cephoid. A cephoid is a special convex compact polyhedron: the Minkowski sum of a finite set of (“positive”) simplices. We aim for the computation of the maximal faces of such polyhedra. Within the presentation we will also deal with other data as well, e.g., the normal of a maximal face, the dual version of a cephoid, the reference vector of a maximal face, etc.
1 Introduction, Faces

A cephoid is an algebraic ("Minkowski") sum of finitely many simplices. Each of these simplices we assume to be spanned by the vector \( \mathbf{0} \in \mathbb{R}^n \) and \( n \) additional positive vectors located on the axes of \( \mathbb{R}^n \). Thus we speak of "deGua"–simplices in view of J.P. de Gua de Malves [1] who first discussed a generalization of the Pythagorean Theorem, thereby dealing with a positive simplex as described above.

To make this more precise, let \( \mathbf{a} = (a_1, \ldots, a_n) > 0 \in \mathbb{R}^n \), be a positive vector. Let \( I := \{1, \ldots, n\} \) and denote by \( \mathbf{e}^i \) be the \( i \)th unit vector \( (i \in I) \). Also, let \( \mathbf{a}^i := a_i \mathbf{e}^i \ (i \in I := \{1, \ldots, n\}) \) denote the multiple of \( \mathbf{e}^i \) indicated by the \( i \)th coordinate of \( \mathbf{a} \).

Then the vectors \( \mathbf{a}^i \ (i \in I) \) together with the vector \( \mathbf{0} \) span a simplex (the convex hull of these vectors), which is written \( \Pi^\mathbf{a} := \text{conv}\{\mathbf{0}, \mathbf{a}^1, \ldots, \mathbf{a}^n\} \) (conv denoting the convex hull). Now \( \Pi^\mathbf{a} \) is called a deGua simplex.

Various subsimplices of a deGua simplex are of special interest. We denote by \( \Delta^\mathbf{a} := \text{conv}\{\mathbf{a}^1, \ldots, \mathbf{a}^n\} \) without the vector \( \mathbf{0} \), i.e., this is the outward surface of \( \Pi^\mathbf{a} \). Also, for \( \emptyset \neq J \subseteq I \), we write \( \Delta^\mathbf{a}_J := \text{conv}\{\mathbf{a}^i \mid i \in J\} \) thus obtaining the subsimplex of \( \Delta^\mathbf{a} \) corresponding to some subset \( J \) of the index set \( I \).

Next, let \( \mathbf{a}^* = (a^{(k)})_{k=1}^K \) be be a family of positive vectors in \( \mathbb{R}^n \). We write \( K := \{1, \ldots, K\} \). Then the algebraic or Minkowski sum

\[
\Pi = \Pi^{\mathbf{a}^*} := \sum_{k=1}^K \Pi^{\mathbf{a}^{(k)}} = \sum_{k \in K} \Pi^{(k)}
\]

is called a cephoid. Cephoids have been introduced in [5], see also [2],[4]. Throughout this paper we assume that a cephoid is nondegenerate, see [5] for the details.

In \( \mathbb{R}^n \) the data of a cephoid are given by \( K \) vectors arranged in a \( K \times N \)–matrix, generically denoted by \( \mathbf{A} \). That is, a cephoid is represented by a \( K \times N \)–array of shape \( \rho = 2 \). For example we have the \( 3 \times 8 \) cephoid \text{BLUE} given via
Note that the array \( A \) denotes the family of vectors, not the algebraic sum – the cephoid – as such. It is our aim to compute the data of the cephoid by means of the appropriate computational tools, that is APL programs.

Now, the outer surface (“Pareto surface”, “cephoidal surface”) of a cephoid \( \Pi \) is denoted by \( \partial \Pi \). In order to describe the maximal faces of \( \partial \Pi \), it is appropriate to recall the “Coincidence Theorem” ([5]). It states the following.

**Theorem 1.1.** Given some maximal face \( F \) of \( \partial \Pi \), there is, for each \( k \in K \) an index set \( J^{(k)} \subseteq I \) and a corresponding subsimplex \( \Delta^{(k)}_{J^{(k)}} \) of \( \Delta^{(k)} \) such that

\[
F = \sum_{k=1}^{K} \Delta^{(k)}_{J^{(k)}}
\]

holds true. Furthermore, there is a set of coefficients (unique up to a positive multiple constant) \( c^* = (c^*_k)_{k \in K} \) such that \( F \) has the same normal as the convex hull of the “adjusted” subfaces \( c^*_k \Delta^{(k)}_{J^{(k)}} \).

In this context we prefer to also use the term “maximum” for the convex hull referring to the partial ordering induced by inclusion on convex sets; hence this convex hull is denoted by

\[
\bigvee_{k \in K} c^*_k \Delta^{(k)}_{J^{(k)}}.
\]
The sets $J^{(k)}$ are called the \textit{reference sets}, the collection $\mathcal{J} = \{J^{(k)}\}_{k \in K}$ is the \textit{reference system} of $F$. The reference system defines $F$ uniquely.

E.g., a face of the cephoid $\text{BLUE}$ called $\text{BLUEF25}$ is given by the reference system

$$J^{(1)} = \{5, 6\}, \quad J^{(2)} = \{1, 3, 4, 7\}, \quad J^{(3)} = \{1, 2, 5\}.$$  

the precise way to write this faces is

$$\text{BLUEF25} := \Delta_{\{56\}} + \Delta_{\{1347\}} + \Delta_{\{125\}},$$

where the vectors $a^{(1)}, a^{(2)}, a^{(3)}$ are given by the rows of the matrix $\text{BLUE}$.

To represent this by an appropriate array we choose a $N \times K$ matrix, the columns of which correspond to the various reference sets. Then we obtain for $\text{BLUEF25}$:

\begin{verbatim}
*****************
BLUEF25
0 1 1
0 0 2
0 3 0
0 4 0
5 0 5
6 0 0
0 7 0
WR BLUEF25
5 6 ⊘ 1 3 4 7 ⊘ 1 2 5 ⊘
*****************
\end{verbatim}

The function $\text{WR}$ writes a face; it helps to obtain an immediate overview about the index sets suggested by the array that is used for computational purpose within $\text{APL}$ context. Within the above sketch we have used the function $\text{WR}$ in order to write the face $\text{BLUEF25}$ in a way which resembles formula (3).

The \textit{adjustment coefficients} as mentioned above $c^*_k$ are determined uniquely up to a positive multiple. The set

$$L := \{l \in I \mid l \text{ appears in at least two of the sets } J^{(k)}\}$$
is called the adjustment set. It serves to determine the normal of $F$ as follows. We write $L^{(k)} := L \cap J^{(k)}$ and
\[(5) \quad \mathbb{L} := \left\{ (k, l) \mid l \in L, J^{(k)} \ni l \right\} = \left\{ (k, l) \mid l \in L^{(k)} \right\}\]
and obtain the linear adjustment system which is the homogeneous linear system of equations in variables $(c_k, \lambda_l)$, $((k, l) \in \mathbb{L})$, given by
\[c_k a_i^{(k)} = \lambda_l \quad ((k, l) \in \mathbb{L}).\]

As shown in [5], this system (for fixed $F$) has a unique solution (up to a positive constant) $(c_\star, \lambda_\star)$, the first ingredients of which yield the adjustment coefficients. Moreover, the normal $n_\star$ of $F$ is obtained by computing
\[\text{normalpermax} \quad (7) \quad a_i^\star := \max_{k \in K} c_k a_i^{(k)} \quad (i \in I)\]
and
\[\text{normalpermax2} \quad (8) \quad n^\star = \left( \frac{1}{a_1^\star}, \ldots, \frac{1}{a_n^\star} \right).\]

Finally, note that the linear functional $x \mapsto n^\star x$ attains its maximum restricted to $\Pi^{(k)} = \Pi^{a^{(k)}}$ exactly on $\Delta^{(k)}$. “Adjustment” means that the maximal value of this functional relative to $c_k^\star \Pi^{(k)}$ (which is attained exactly on $c_k^\star \Delta^{(k)}$) equals some common value $t^\star$ for all $k \in K$ – which is why $n^\star$ is indeed the normal to
\[\bigvee_{k \in K} c_k^\star \Delta^{(k)}_{J^{(k)}}\]
as well as to $F$.

The computational procedures for to deal with these objects are constructed as follows. The solutions to the linear adjustment system are given by the function \textsc{ADJUSTCOEF}. Note that this is a function of two variables as the face is given via the reference system referring to the matrix $A$ representing a cephoid. Thus we have

\begin{verbatim}
BLUEF25 ADJUSTCOEF BLUE
0.0035982814178302912 0.01818181818181814 0.010204081632653064
\end{verbatim}

Next we determine the coefficient matrix of the linear adjustment system. To this end we rewrite the linear system (9) in the form
\[c_k a_i^{(k)} = \lambda_l \quad ((k, l) \in \mathbb{L}),\]
such that the coefficients of the $\lambda_\rho$ are all equal to $-1$. The program \textsc{CO-EFFLINADJ} then yields for instance:
Based on these procedures we are now in the position to compute the normal to a face. In order to do so we solve the linear adjustment system and thereafter just take the inverse vector ("coordinatewise"). The result is our first basic function: $\Delta$NORMAL which we represent fully as follows:

\[ [0] \text{ NOR } F \Delta\text{NORMAL A};C;CM;CSTAR \]
\[ [1] \]
\[ [2] \text{ computes the normal to a face } F \text{ of a cephoid } A. \]
\[ [3] \text{ uses the linear adjustment system, thus computes } \]
\[ [4] \text{ the solution to the linear adjustment system, then inverts } \]
\[ [5] \text{ the vector coordinatewise. } \]
\[ [6] \text{ used as subroutines are therefore functions } \]
Section 1: Faces

[ 8] ⋆ ADJUSTCOEFF ⋆, ⋆ COEFFLINADJ ⋆, ⋆ LMATRIX ⋆ AND
[ 9] ⋆ THEIR SUBROUTINES.
[10] ⋆ ******************************************************************************
[11] ⋆
[12] C+ F ADJUSTCOEF A
[13] →(C=0)/ERROR
[14] ⋆ C+0 1 2
[15] CM+ (((pA)[2]) / ((pC),1)ρ C)×A
[16] CSTAR ← /[1] CM
[17] NOR+1=CSTAR
[18] →0
[19] ⋆ FFFFFFFFFFFFFFFFFFFFFFFFFFFFF
[20] ERROR:
[21] 'ERROR WHEN COMPUTING A NORMAL VIA ⋆ NORMALF ⋆ !'
[22] 'NORMAL IS ISSUED WITH 0.'
[23] NOR ← 0
2 The Dual Cephoid

To any cephoid represented by a matrix $A$ we can construct its dual by just taking the adjoint or transposed matrix $A^T$. Assume that $A$ is a $K \times N$-matrix, that is the cephoid

$$\Pi = \sum_{k=1}^{K} \Pi^{a(k)}$$

is a sum of $K$ deGua simplices in $\mathbb{R}^n$. Then, clearly the dual cephoid is a sum of $n$ simplices in $\mathbb{R}^K$. We write $\tilde{a}^{(i)}_k := a^{(k)}_i (i \in I, k \in K)$, and call the family

$$\left( \tilde{a}^{(i)}_k \right)_{i \in I}$$

the dual family. The cephoid

$$\Pi^* = \Pi^{\tilde{a}^*} = \sum_{i \in I} \Pi^{\tilde{a}^{(i)}}$$

is the dual cephoid.

The dual cephoid is at once seen to be given by $QA$. It is the nice result of [3] that the faces of the dual cephoid are obtained by simple operation from the faces of the primal cephoid. To be precise, let $F$ be a maximal face of $\Pi$ given by its reference system $J = \left( J^{(k)} \right)_{k \in K}$. For $i \in I$ we write

$$J^{(i)} := \left\{ k \in K \mid i \in J^{(k)} \right\},$$

then it turns out that

$$\tilde{J} = \left( \tilde{J}^{(i)} \right)_{i \in I}$$

(called the dual reference system) defines a maximal face of $\Pi$. This way a bijective mapping between the faces of both the primal and dual cephoid is established. The dual face to some face $F$ is, therefore, easily computed, we use function DUALFACE given as follows.
Obviously, this operation can be performed with a family of faces, in particular with the family of all faces of Π; this is done by the function \( \text{DUALFAM} \).

For example, turning to the 25\(^{th}\) face of cephoid BLUE we obtain

```
BLUEF25
0 1 1
0 0 2
0 3 0
0 4 0
5 0 5
6 0 0
0 7 0

WR BLUEF25
5 6 1 3 4 7 1 2 5
```

```
**********

DUALFACE BLUEF25
0 0 0 0 1 1 0
2 0 2 2 0 0 2
3 3 0 0 3 0 0

WR DUALFACE BLUEF25
2 3 1 3 2 2 1 3 1 2
```

Now, the faces of the dual cephoid obey the same laws as the one of the primal
cephoid. In particular, we can construct the linear adjustment system of the dual $\bar{F}$ to a face $F$. The dual adjustment set is then

$$\bar{L} := \left\{ k \in K \bigg| k \text{ is in at least two different } \bar{J}^{(i)} \right\}$$

$$= \left\{ k \in K \bigg| |J^{(k)}| \geq 2 \right\}$$

which yields the set

$$\bar{\mathbb{L}} := \left\{ (i,s) \bigg| s \in \bar{L}, \ J^{(i)} \ni s \right\} = \left\{ (i,s) \bigg| s \in \bar{L}^{(i)} \right\}$$

$$= \left\{ (i,s) \bigg| i \in J^{(s)}, \ |J^{(s)}| \geq 2 \right\}.$$ analogously to (9) of Section 1. Hence the dual linear adjustment system is the linear system of equations in variables $(n_*, \mu_*)$

$$\text{duallinsyst} \ (7) \quad \pi_s^{(i)} n_i = \mu_s \ ((i,s) \in \bar{\mathbb{L}}).$$

Now, in [3] it is shown that the adjustment coefficients of the primal face constitute the normal of the dual face and vice versa. In particular, the system (7) directly serves to compute the normal of the primal face.

Based on this result we obtain a second method of computing the normal to a face of a cephoid.
When comparing both methods and the results obtained by computing the normals via the two programs obtained, a caveat is in order. Both the linear adjustment systems are homogeneous systems of degree 1, that is, the solution space is one-dimensional (i.e. the solutions are just unique up to a multiple constant – so is the normal to a face).

We obtain a solution by requiring a variable to satisfy \( \lambda_1 = 1 \) within the system. But the transposition of the matrix and the dual face operation involved in program \texttt{ANORMALD} may change this normalizations, so both results may differ. For example, we seem to have a discrepancy in the following computations, but it turns out that the first result is a multiple of the second.

\[
\begin{array}{llllllllllllllll}
0.99999999999999984 & 0.87499999999999968 & 0.71428571428571416 \\
0.55555555555555544 & 1.4626865671641786 \\
1.323383084577114 & 1.122489795918366
\end{array}
\]

\[
\begin{array}{llllllllllllllll}
0.0035982814178302904 & 0.0031484962406015032 & 0.0025702010127359212 \\
0.0019990452321279384 & 0.0052631578947368432 \\
0.0047619047619047632 & 0.0040388873057278768
\end{array}
\]

\[
\begin{array}{llllllllllllllll}
0.0035982814178302912 & 0.0035982814178302908 & 0.00359828141783029 \\
0.00359828141783029 & 0.0035982814178302916 \\
0.003598281417830292 & 0.0035982814178302904
\end{array}
\]

\[
\begin{array}{llllllllllllllll}
0.0035982814178302912 & 0.0035982814178302908 & 0.00359828141783029 \\
0.00359828141783029 & 0.0035982814178302916 \\
0.003598281417830292 & 0.0035982814178302904
\end{array}
\]
3 The First Algorithm

Within this section we describe and implement an algorithm for computing all maximal faces of a (non–degenerate) cephoid. To this end, we employ a recursive procedure describing the number and nature of maximal faces of such a cephoid \( \Pi = \sum_{k \in K} \). The procedure – and hence the idea of our algorithm – in this case is derived from [5]. We shall use the fact that, for \( n \geq K \) every maximal face of a cephoid \( \Pi \) can be found on the \( n-1 \) boundaries, i.e., has a nonempty intersection with the outward surface of one of the lower dimensional cephoids \( \Pi_{I\setminus\{i\}} \) for some \( i \in I \).

Let \( J \subseteq I \). We say that a maximal face \( F \) of \( \Pi \) has a proper \( J \)-cut if

\[
\dim (F \cap \mathbb{R}^n_{J+}) = |J| - 1
\]

holds true, i.e., the intersection has the dimension of the \( \mathbb{R}^n_{J+} \)-boundary of the (outer) surface \( \partial \Pi \) of \( \Pi \). Another way of interpreting this definition is as follows. Consider, for any \( a^{(k)} \) the restriction to \( \mathbb{R}^n_{J} \) called \( a^{(k)}_J \). Then

\[
\Pi_J = \sum_{k \in K} \Pi^{a^{(k)}}_J \subset \mathbb{R}^n_{J+}
\]

is a cephoid in \( \mathbb{R}^n_{J+} \). It is not hard to see that \( F \) has a proper \( J \)-cut if and only if

\[
F \cap \mathbb{R}^n_{J+}
\]

is a maximal face of \( \Pi_J \).

Proper cuts are ordered by inclusion. A minimal proper cut is a proper cut with minimal dimension, that is with minimal size of \( J \). We call \( F \) an \( l \)-based face if the dimension of the minimal proper cut is \( l - 1 \).

Thus, a 1–based face contains a vertex, a 2–based face cuts properly into a 2 dimensional subcephoid of \( \Pi \), etc.

Now, based on Proposition 6.1 of [5] we formulate the following remark.

**Lemma 3.1.** Let \( F \) be a maximal face of \( \Pi \). Then \( F \) is an \( l \)-based face for some \( l \leq \min\{K - 1, n\} \). The set \( J \) that yields the minimal proper cut is exactly the set \( L \) of reference indices of \( F \). In particular, if \( K \leq n \) holds true, then any maximal face is \( l \) based with some \( l \leq n-1 \).

As a consequence, for \( K \leq n \) any maximal face of a cephoid \( \Pi \) induces a maximal face of a lower dimensional cephoid of dimension of at most \( K-1 \leq \)
This lower dimensional maximal face however has the same set of reference indices \( L \). As the reference indices uniquely determine a maximal face, we can identify the original face by its lower dimensional induced version.

This fact clearly hints to a recursive procedure in order to compute all maximal faces of a cephoid: we look to the maximal faces of \((n - 1)\)-dimensional subcephoids and try to augment the corresponding reference sets accordingly.

This augmentation is performed by means of the linear adjustment system. For, given the set \( L \) (which stems from a face \( F^{(-i)} \) of some \((n - 1)\)-dimensional cephoid \( \Pi^{(-i)} \)), we can compute the normal, say \( n^* \), to the corresponding face \( F \) in \( n \) dimensions. Now, given this normal, the reference system of the face is obtained at once. For, the reference set \( J^{(k)} \) of \( F \) is described by the fact that the functional \( x \mapsto n^* x \) attains its maximum relatively to \( \Delta^{(k)} \) exactly on \( \Delta^{(k)} \). Therefore, given \( n^* \), the set \( J^{(k)} \) consists of just those \( i \in I \) that maximize the expression \( a^{(k)} n^* = a_i^{(k)} n_i^* \).

Let us describe a more formal approach as follows.

We write

\[
(2) \quad a^* := (a^{(k)})_{\kappa \in K}
\]

for a family of positive vectors generating a cephoid. Consider the mapping

\[
(3) \quad \mathcal{F}(K, n; \ast) : \left\{ a^* := (a^{(k)})_{\kappa \in K} \mid a^* \text{ is non-degenerate} \right\} \rightarrow \mathcal{P}\left((\mathcal{P}(I))^K\right)
\]

(where \( \mathcal{P}(\bullet) \) denotes the power set) which associates with a set of positive vectors in \( \mathbb{R}_+^n \) a finite set \( \mathcal{F}(K, n; a^*) \) of reference systems \((J^{(1)}, \ldots, J^{(K)})\) corresponding to the faces of \( \Pi = \sum_{i \in K} \Pi^{(k)} \). We base an algorithm on the following Theorem that is proved in [5]:

**Theorem 3.2.** The function \( \mathcal{F}(K, n; \ast) \) can be recursively computed using the functions \( \mathcal{F}(K', n; \ast) \) \((K' < K)\) and the functions \( \mathcal{F}(K, n'; \ast) \) \((n' < n)\).

**Proof:**

1st STEP: We start with \( n = 2 \). In this case the cephoidal surface consists of line segments. For arbitrary \( K \) assume that the slopes of line segments \( \frac{a_i^{(k)}}{a_2^{(k)}} \) are strictly decreasing in \( k \). Then the faces of \( \Pi \) are given by

\[
(4) \quad F^{(k)} := \sum_{l<k} a^{(l)1} + \Delta a^{(k)} + \sum_{l>k} a^{(l)2}.
\]
Equivalently, the corresponding index sets are given by
\[
J^{(1)} = \ldots = J^{(k-1)} = \{1\},
\]
\[
J^{(k)} = \{1, 2\},
\]
\[
J^{(k+1)} = \ldots = J^{(K)} = \{2\}.
\]

The corresponding matrix representing such a face in our code is given by
\[
\begin{pmatrix}
1 \cdots 1 & 1 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 2 & 2 & \cdots & 2
\end{pmatrix}.
\]

Thus, in this situation, the maximal faces are just the translates of the line segments and the computation is rather straightforward.

In general, as the line segments involved will not be ordered according to decreasing slopes, one has to employ a suitable permutation generating the correct ordering.

This way, all faces are completely described in case that \( n = 2 \) holds true; an APL-function representing this algorithm is \( \text{FACN2} \):

\[
\text{FAM} + \text{FACN2} A;K;N;L;Q;UPTRI;LOWTRI
\]

1. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
2. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
3. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
4. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
5. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
6. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
7. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
8. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
9. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
10. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
11. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
12. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
13. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
14. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
15. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
16. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
17. \( \text{FAM} \) + \( \text{FACN2} \) \( A;K;N;L;Q;UPTRI;LOWTRI \)
Section 3: 1st Algorithm

[18] \( \ast \) TAKING A FACE BY REARRANGING THE COLUMNS OF BOTH TRIANGLES:
[19] \( \ast \)
[20] \( L+0 \)
[21] \( \text{FAM} \leftarrow 0p0 \)
[22] \( \ast \) COUNTING THE COLUMNS:
[23] \( \ast \) -----------------------------------------------
[24] \( \text{UP} \leftarrow (K<L \leftarrow L+1)/0 \)
[25] \( \text{FAM} \leftarrow \text{FAM}, c(2,K)p(\text{UPTRI}[L;][PI]),(\text{LOWTRI}[L;])[PI] \)
[26] \( \rightarrow \text{UP} \)
[27] \( \text{WARN:} \)
[28] 'N IS NOT 2 IN PROGRAM */FACN2*/'
[29] 'FAMILY IS ISSUED WITH 0p0'
[30] \( \text{FAM} \leftarrow 0p0 \)

As an example we compute the faces of two columns of the transposed matrix of BLUE:

\[
\begin{array}{llllllllllll}
110 & 130 & 150 & 170 & 190 & 210 & 230 \\
55 & 33 & 77 & 99 & 11 & 1 & 49 \\
98 & 112 & 133 & 14 & 67 & 44 & 23 \\
\end{array}
\]

\((\& \text{BLUE })[;1 \ 3]\)

\[
\begin{array}{llllllllllll}
110 & 98 \\
130 & 112 \\
150 & 133 \\
170 & 14 \\
190 & 67 \\
210 & 44 \\
230 & 23 \\
\end{array}
\]

\(\Delta \text{FACN2} (\& \text{BLUE })[;1 \ 3]\)

\[
\begin{array}{llllllllllll}
1 & 2 & \circ & \circ & 1 & \circ & 1 & \circ & 1 & \circ & 1 & \circ \\
2 & \circ & 1 & \circ & 1 & 2 & \circ & 1 & \circ & 1 & \circ & 1 \\
2 & \circ & 1 & 2 & \circ & 2 & \circ & 1 & \circ & 1 & \circ & 1 \\
2 & \circ & 2 & \circ & 2 & \circ & 1 & \circ & 2 & \circ & 1 & \circ \\
2 & \circ & 2 & \circ & 2 & \circ & 1 & \circ & 2 & \circ & 1 & \circ \\
\end{array}
\]
2ndSTEP: For completeness we consider the case $K = 2$ for dimension $n \geq 3$ that has been treated in Section 4 of [5]. We do not have to develop a separate program for this case if we are willing to employ duality theory. However, we want to shortly describe the “duality free” procedure.

In case of two cephoids we call the deGua simplices involved $\Pi^a$ and $\Pi^b$. Now, for any two prisms $\Pi^a$ and $\Pi^b$, there is an ordering $\prec$ of $I$ that completely describes all faces via

$$F^\prec := \Delta^a_{S^\prec} + \Delta^b_{T^\prec} \quad (i \in I).$$

with

$$T^\prec_j := \{i \in I \mid i \prec j\} \cup \{j\} \quad S^\prec_j := \{i \in I \mid j \prec i\} \cup \{j\}$$

denoting the predecessors (resp. successors) of $j \in I$ including $j$.

Now, for $n = 3$, determine $i \in I = \{1, 2, 3\}$ such that $b^i$ admits the normal of $\Delta^a$. Next, determine $j \in I = \{1, 2, 3\}$ such that $a^j$ admits the normal of $\Delta^b$. Let $k$ be the third index. Then the ordering is $i \prec k \prec j$.

For $n > 3$, suppose the procedure is known for $n - 1$. Compute the ordering on $I \setminus \{n\}$ by projecting all prisms and the sum in $\mathbb{R}^n_{I \setminus \{n\}}$. Likewise proceed in order to obtain an ordering on $I \setminus \{1\}$. The orderings are necessarily consistent and define an ordering on $I$. This generates all faces and index sets via (6).

Here is a corresponding program. Note that the procedure includes more recursion, hence is less “APL”–elegant but possibly faster ....
[ 6]  ₪ THUS CONSISTS OF TWO ROWS.
[ 7]  ₪ ******************************************
[ 8]
[ 9]  K+(rA)[2]
[11]
[12]  ₪ ......................................................
[13]  ₪ FACES ARE CONSTRUCTED TAKING INTO ACCOUNT
[14]  ₪ THE ORDERING OR PERMUTATION
[16]  ₪ STRAIGHT LINES CORRESPONDING TO THE COLUMNS OF A
[17]  ₪ ........................................................
[18]  PI ← 4Q
[19]  ₪P
[20]  ₪'
[21]  ₪ WE MAY LIST THE PERMUTATION IN CASE IT IS NEEDED:'
[22]  ₪Q
[23]  ₪ ......................................................
[24]  L+1 = M+Q(2,K)p(uK),Kp0
[25]  M[PI[1];]+ (PI[1],PI[1])
[26]
[27]  FAM ← M
[28]  ₪  ........................................................
[29]  UP:
[30]  →(K<L+L+1)/ END
[31]  M[PI[L-1];]+ (0,PI[L-1])
[32]  M[PI[L];]+ (PI[L],PI[L])
[33]  FAM=FAM, c M
[34]  →UP
[35]  END:
3rd STEP:
Now we proceed by induction in $K$ and $n$. For $K > n$, we may use duality in order to convert our problem to a situation in which $K < n$ holds true. We may therefore assume $K \leq n$ to be the case.

4th STEP: If $K \leq n$ then, for any maximal face $F$, we know that $F$ is an $l$-based face with $l \leq K - 1 \leq n - 1$. Consider the representation by the reference system, say

\[ F = \sum_{k=1}^{K} \Delta^{(k)}_{j(k)} \tag{8} \]

and let $L \subseteq I$ with $|L| = l$ be the set of adjustment indices. As $l \leq n - 1$ is true, there is at least one index in $I$ that is not an element of $L$. Assume for the sake of simplicity that this index is $n$.

Consider the projection on $\mathbb{R}^{n}_{I \setminus \{n\}}$. Then we have $\Delta^{(k)}_{j(k)} \cap \mathbb{R}^{n}_{I \setminus \{n\}} = \Delta^{(k)}_{j(k) \setminus \{n\}}$.

As $F_{I \setminus \{n\}}$ is maximal with respect to $\Pi_{I \setminus \{n\}} = \sum_{k=1}^{K} \Delta^{(k)}_{I \setminus \{n\}}$ we note that

\[ F_{I \setminus \{n\}} = \sum_{k=1}^{K} \Delta^{(k)}_{j(k) \setminus \{n\}}. \tag{9} \]

Hence, $F_{I \setminus \{n\}}$ is an $l$-based face with the same set of boundary indices $L$. As the dimension has been reduced by one, $F_{I \setminus \{n\}}$ or rather the reference system can be computed using the procedures defining the functions $\mathcal{F}(K, n'; \star)$ for $n' < n$.

5th STEP:
Now, the set $L$ of adjustment indices obtained for some face $F_{I \setminus \{n\}}$ recursively is as well the adjustment set for the face $F$. The set $L$ determines the face $F$ uniquely and the way to handle this computationally is indicated by the adjustment system. The adjustment system can be set up (in full $n$ dimensions) and yields the normal of $F$. The normal in turn determines the extremals of $F$ as, for any $k \in K$, these are those vectors $a^{(k)i}$ ($i \in I$) that maximize the inner product with the normal relative to $\Delta^{(k)}$.

6th STEP: We have to observe that the transition from $F$ to $F_{I \setminus \{n\}}$ is not unique. In other words, a face $F$ may have quite a few descendants in $n - 1$ dimensions or less, thus appear several times when the recursion from lower dimensions to $n$ is enacted. To prevent multiple listings, we have to eliminate double appearances of faces. A possible procedure is to appeal to
the reference vectors (and not to the reference sets) as these provide less data to check through. The reference vector determines a face uniquely, hence it suffices to carry on a list of reference vectors and delete multiple listings accordingly.

We provide the Function ΔFCR that incorporates the algorithm described above.

```plaintext
[ 0] FACGLOB ← ΔFCR A;K;N;D;RVGLOB;K1;RVGL11;A1;FAM1;REFM1;K2;F1;I0;F;R1
[ 1] ⢬
[ 2] ⢬ ******************************************************************************
[ 3] ⢬ GENERATES ALL FACES OF A CEPHOID REPRESENTED BY THE
[ 5] ⢬ THE MATRIX OF REFERENCE VECTORS.
[ 6] ⢬ ******************************************************************************
[ 8] ⢬ OF WHICH IS AN INCLUDED N × K MATRIX REPRESENTING A FACE.
[ 9] ⢬ ******************************************************************************
[10] ⢬
[11] ⢬DATA:
[12] K ← (ρA)[1]
[13] N ← (ρA)[2]
[14] ⢬ K = 2 OR N=2: REDUCTION TO PROGRAMS ACCORDINGLY.
[15] ⢬ ******************************************************************************
[16] →(K>2)/CONS
[17] FACGLOB ← ΔFACK2NBEL A
[18] →0
[19] ⢬ ******************************************************************************
[20] CONS:
[21] ⢬
[22] →(N>2)/CONN
[23] FACGLOB ← ΔFACN2KBEL
[24] →0
[25] ⢬ ******************************************************************************
[26] CONN:
[27] ⢬ IF N EXCEEDS K, THEN USE THE DUAL VERSION:
[28] →(N≤K)/PRIMAL
[29] FACGLOB ← DUALFAM ΔFCR ⨿ A
```
[30] \rightarrow 0
[31] \star .............................................................
[32] PRIMAL:
[33] FACGLOB + 0p1
[34] \star THE MATRIX OF REFERENCE VECTORS (K,N):
[35] RVGLOB + K ALLREFVECS N
[36] .................................................................
[37] \star BEGIN OF ROUTINE:
[38] \star INITIATE ROWS OF RVGLOB:
[39] K1+0
[40] \rightarrow ( K < K1,K1+1)/0
[41] \star ROWS OF RVGLOB WITH ENTRY 1 AT K1:
[42] .................................................................
[43] RVGL11 + (RVGLOB[;K1]=1)/[1] RVGLOB
[44] \rightarrow (0 = (pRVGL11)[1])/UP
[45] \star
[46] \star DETERMINE FACES OF THE CEPHOID OBTAINED BY A^(-K1),
[47] \star ITS FACES AND THE CORRESPONDING REFVECTORS.
[48] \star THIS CONTAINS INDUCTIONSTEP ... 
[49] A1 + (1 - K1 = \lambda K)/[1]A
[50] FAM1 + \Delta FCR A1
[51] REFM1 + REFVECMAT FAM1
[52] \star
[53] \star +==================================================================
[54] \star INITIATE ROWS OF RVGL11
[55] \star
[56] K2+0
[57] \rightarrow ( pRVGL11)[1] < K2 + K2+1)/DELET
[58] \star REDUCE THE ROW OF RVGL11 AT HAND AT K1:
[59] R1 + RVGL11[K2;]
[60] R1 + (1 - K1 = \lambda K)/R1
[61] \star AND CONSIDER THE FACE GENERATED FOR THIS VECTOR
[62] \star WRT THE REDUCED MATRIX A1:
[63] F1 + FAM1 SEEKFACE REFM1,[1] R1
[64] \star LOOK FOR THE OPTIMAL VERTEX OF A^(K1) RELATIVE TO F1:
[66] \star EXTEND F1 ABD ADD TO THE FAMILY IN CONSTRUCTION:
[67] F + (K1,IO) SUBEXTF F1
[68] FACGLOB + FACGLOB,[1] c F
[69] \rightarrow UP
[70] \star
Section 3: 1st Algorithm

[71] \[ \alpha \]

[72] DELET:

[73] RVGLOB \leftarrow K1 \text{ CANCL RVGLOB}

[74] \rightarrow \text{UP}

q.e.d.
4 The Second Algorithm

This algorithm is historically based on the first procedure for enumerating all faces of a cephoid, see Section 6 of [5]. This original version does not make use of duality theory, it is fully located in the primal domain.

We distinguish two cases as follows.

Assume first of all $K < n$. Then any maximal face is $l$–based with $l \leq (K - 1) \leq (n - 2)$. That is, a maximal face is essentially determined by a boundary face with at least two dimensions less. This gives rise to a recursive procedure: we compute all those boundary faces and extend them to the full dimension. The matter of extension is also different from the one employed in our first algorithm: for a face of an $(n - 2)$–dimensional sub–cephoid is it seen that the extension is a rather direct procedure.

Assume then that $K \geq n$ is the case. According to Corollary 6.8 in [5] we know that in this case any maximal face of a cephoid has at most $(n - 1) < K$ nontrivial reference sets. Thus, we can compute the maximal faces for a sum of $K_0 = n - 1$ cephoids. The remaining summands correspond to one dimensional reference sets, in other words, they are vertices of the various $\Delta^{(k)}$. These vertices are uniquely determined according to Theorem 4.2 of [5].

We attempt to describe this step of the algorithm in more detail following the 4th step of the proof of Theorem 6.2 in [5].

So we assume $K \leq n - 1$. Let $F$ be a maximal face, we know that $F$ is an $l$–based face with $l \leq K - 1 \leq n - 2$. Consider the representation by the reference system, say

\begin{equation}
F = \sum_{k=1}^{K} \Delta_{j(k)}^{(k)}
\end{equation}

and let $L \subseteq I$ with $|L| = l$ be the set of adjustment indices. As $l \leq n - 2$ is true, there are at least two indices in $I$ that are not elements of $L$. Assume for the sake of simplicity that these indices are 1, $n$.

Now, first of all, consider the projection on $\mathbb{R}^n_{I \setminus \{n\}}$. Then we have $\Delta_{j(k)}^{(k)} \cap \mathbb{R}^n_{I \setminus \{n\}} = \Delta_{j(k \setminus \{n\})}^{(k)}$. As $F_{I \setminus \{n\}}$ is maximal with respect to $\Pi_{I \setminus \{n\}} = \sum_{k=1}^{K} \Delta_{I \setminus \{n\}}^{(k)}$ we note that

\begin{equation}
F_{I \setminus \{n\}} = \sum_{k=1}^{K} \Delta_{j(k \setminus \{n\})}^{(k)}.
\end{equation}
Hence, \( F_{I\setminus\{n\}} \) is an \( l \)-based face with the same set of boundary indices \( L \). Now \( F_{I\setminus\{n\}} \) or rather the reference system can be computed using the procedures defining the functions \( \mathcal{F}(K, n'; \star) \) for \( n' < n \). This is so as the dimension has been reduced properly.

The same procedure can now be followed using the projection on \( \mathbb{R}^n_{I\setminus\{1\}} \).

Then, for any \( k \) we have computed recursively the index sets

\[
J^{(k)} \setminus \{n\}, \ J^{(k)} \setminus \{1\},
\]

and as

\[
J^{(k)} = (J^{(k)} \setminus \{n\}) \cup (J^{(k)} \setminus \{1\}).
\]

Instead of two indices this procedure can be performed with any finite number of indexes not in \( L \).

Thus, for any \( k \in K \), the index set \( J^{(k)} \) is determined via

\[
(3) \quad J^{(k)} = \bigcup_{i \notin L} \left( J^{(k)} \setminus \{i\} \right)
\]

which defines the face \( F \) of \( \Pi \) or an element of \( \mathcal{F}(K, n; a^\star) \).
5 The Third Algorithm

Our third algorithm results from the Bijection Theorem regarding faces and reference vectors of a cephoid, see Theorem 3.2 as well as Corollaries 3.3. and 3.4 of [3]. Accordingly, we seek to construct all faces by running through a list of the reference vectors and computing to each such reference vector the appropriate face.

The procedure is recursive as well. It is based on the idea that, for \( n \leq K \), a reference vector has at least one coordinate that equals 1. Indeed, according to the definition, a reference vector \( \mathbf{r} = (r_1, \ldots, r_K) \) has to satisfy \( r_k \leq n \ (k \in K) \) and

\[
\sum_{k=1}^{K} r_k = K + n - 1.
\]

Now, if \( r_k \geq 2 \) holds true for all \( k \in K \), then we have

\[
n + K - 1 = \sum_{k \in K} r_k \geq 2K,
\]

that is, \( n - 1 \geq K \).

Hence we are going to deal with the case \( n \leq K \) and the alternative \( n > K \) is treated via an application of Duality Theorem as in the First Algorithm.

Now, consider the case that \( r_K = 1 \). Then we observe that the vector \((r_1, \ldots, r_{(K-1)})\) is a reference vector for the cephoid \( \Pi^0 := \sum_{k \in K \setminus \{K\}} \Delta^{(k)} \).

in view of

\[
\sum_{k \in K \setminus \{K\}} r_k = \sum_{k \in K} r_k - 1 = (n + K - 1) - 1 = n + (K - 1) - 1.
\]

Therefore, we can obtain the maximal face \( F^0 \) of \( \Pi^0 \) corresponding to this reference vector by a recursive procedure as \( \Pi^0 \) is a sum of \( K - 1 \) deGua simplices in \( \mathbb{R}^n \). It remains to construct the maximal face of \( \Pi \) that results from adding a (unique) vertex \( a^{(K)_0} \) of \( \Delta^{(K)} \) to \( F^0 \).

Now, this vertex does indeed exist uniquely as is stated in Theorem 4.2 and Corollary 4.4 of [2]. There it is shown that, given the normal \( \mathbf{n}^0 \) of \( F^0 \), there is exactly one vertex \( a^{(K)_{io}} \) of \( \Delta^{(K)} \) that admits of this normal, i.e., contains \( \mathbf{n}^0 \) within its normal cone. Of course, this is the unique vertex \( a^{(K)_{io}} \) that maximizes the inner product with \( \mathbf{n}^0 \) over all vertices of \( \Delta^{(K)} \).

Therefore, adding \( a^{(K)_{io}} \) to \( F^0 \) yields a maximal face of \( \Pi \) which has the correct reference vector \( \mathbf{r} \). This describes our third procedure for enumerating all faces of a cephoid.
We shall implement this procedure in APL – code as follows.

1stSTEP: If \( n = 2 \) or \( K = 2 \) is the case then treat this case directly by programs \( \texttt{VACN}2 \) and \( \texttt{VACN}2\texttt{BEL} \), \( \texttt{VACK2BEL} \) respectively.

2ndSTEP: If \( n - 1 \geq K \) is the case, apply dualization twice, i.e., essentially treat the case \( n \leq K - 1 \).

3rdSTEP: Henceforth assume \( n \leq K \).

4thSTEP: Given the family \( a(\bullet) = \{a(k)\}_{k \in K} \); take any subset of \( k - 1 \) elements of this family. Consider e.g. \( \{1, \ldots, (K - 1)\} \) for the sake of the argument. Then, by recursion compute all faces of the family \( \{a(k)_{k \in K - \{K\}}\} \).

To any such face \( F^{(0)} \), compute the normal \( n^{(0)} \). Find the unique vertex \( a^{(i_0)} \) of \( \Delta^{(K)} \) which admits \( n^{(0)} \) as a normal. Then \( F^{(0)} + a^{(i_0)} \) is a face of the original family \( a(\bullet) \).

A more detailed description is provided by the following matrix.

\[
\begin{array}{c}
\text{WS} \star \text{CEPHALGR} \star \\
\text{WS TO COMPUTE ALL FACES OF A CEPHOID ACCORDING TO THE THIRD ALGORITHM WHICH IS BASED ON RECURSION ALONG REFERENCE VECTORS.} \\
\text{**********CREATED DECEMBER 2010 / JANUAR2011} \\
\text{**********} \\
\text{MATRIX} \star \text{INFO} \star \\
\end{array}
\]

A CEPHOID IS REPRESENTED BY A K\times N MATRIX. THE ROWS OF THIS MATRIX CORRESPOND TO THE DE GUÁ SIMPLICES INVOLVED. THAT IS, \( A[I,:] \) REFLECTS A SIMPLEX SPANNED BY VECTORS \( A^\wedge(K) = [0, \ldots, A[K,I], 0 \ldots, 0] \) \((I=1, \ldots, N)\) IN \( R^N_+ \) FOR \( K=1, \ldots, KK \).

EXAMPLE: CEPHOID \( \texttt{BLUE} \) IS A SUM OF 3 VECTORS IN \( R^8 \)

\[
\text{BLUE} \\
110 130 150 170 190 210 230 \\
55 33 77 99 11 1 49 \\
98 112 133 14 67 44 23
\]
A face of a cephoid is represented by a \( n \times k \) matrix. This matrix represents the reference family of the face, each row corresponding to an index \( i \in I \) and each column pointing to a De Gua simplex \( k \). Thus, column \( k \) reflects the index set \( J^k \) \( \subset I \) of the reference family describing the face.

Example: BLUE7 is the sum of two unit vectors \( e^4 \) and the full simplex delta\(^{3}\).

```
BLUE7
0 0 1
0 0 2
0 0 3
4 4 4
0 0 5
0 0 6
0 0 7
```

Faces can be visualized by function \( \ast \) WR \( \ast \) which writes the reference family in a more direct way.

Example:

```
WR BLUE7
4 4 0 1 2 3 4 5 6 7
```

A family of faces is a vector, the coordinates of which are inclusions of the members of that family. Thus, we have \( \rho \ VV134FACEN = 10 \) and \( \rho \ VV134FACEN[1] = \) ZILDE while \( \rho \ VV134FACEN[1] = 3 4 \), as the included 1st coordinate is a 3 4 matrix namely

```
VV134FACEN[1]
1 1 1 1
0 0 0 2
0 0 0 3
```

That is

```
WR VV134FACEN[1]
NOW, THE WHOLE FAMILY CAN BE VISUALIZED VIA THE FUNCTION *WRY* AS FOLLOWS:

```
  WRY VV134FACEN
  1 1 1 1 1 2 3
  1 1 2 1 2 3
  1 1 2 1 2 3
  1 1 2 1 2 3
  1 1 2 1 2 3
```

---

```
FUNCTION *REFVECMAT*
SYNTAX
  RS ← REFVECMAT P

P IS A FAMILY OF FACES; TYPICALLY ALL FACES OF
A CEPHOID, HENCE A DESCRIPTION OF THE SURFACE OF
THAT CEPHOID. THAT IS, P IS A VECTOR THE COORDINATES
OF WHICH ARE INCLUDED N × K MATRICES REPRESENTING
FACES. THE FUNCTION *REFVECMAT* YIELDS THE
CORRESPONDING REFERENCE VECTORS. AS EACH FACE SUPPLIES
A REFERENCE VECTOR, THE RESULT IS A MATRIX THE
ROWS OF WHICH PROVIDE THE REFERENCE VECTORS OF THE
FACES OF THE FAMILY P.

EXAMPLE, REFERRING TO THE ABOVE FAMILY *VV134FACEN*:
  REFVECMAT VV134FACEN
  1 1 1 3
  1 1 2 2
  1 1 3 1
```
1 2 2 1
1 3 1 1
1 2 1 2
3 1 1 1
2 2 1 1
2 1 2 1
2 1 1 2

FUNCTION * ADJUSTCOEF *

SYNTAX:
C ← FACE ADJUSTCOEF A

@****************************************************************************
@ COMPUTES THE ADJUSTMENT COEFFICIENTS OF A FACE *FACE*
@ OF A CEPHOID THAT IS GIVEN BY A MATRIX A.
@*****************************************************************************
SUBROUTINES USED:
   * COEFLINADJ *
   (GENERATES THE COEFFICIENTMATRIX OF THE
    LINEAR ADJUSTMENT SYSTEM),
   * LINEQSOVL *
   (SOLVES THE LINEAR EQUATION AX = B)

EXAMPLE:
   BLUEF7 ADJUSTCOEF BLUE
0.005882352941 0.0101010101 0.07142857143

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

FUNCTION * ΔNORMAL *

SYNTAX:
NOR ← F ΔNORMAL A

@*****************************************************************************
@ COMPUTES THE NORMAL VIA THE LINEAR ADJUSTMENTSYSTEM AND
@ THE ADJUSTMENTCOEFFICIENTS.
Based on Program * AdjustCoef * and the subroutines involved
therein. That is Sub0, Sub1, AdjustInd, LMatrix, LinEqSolv.

Example:

BLUEF7 DNORMAL BLUE
0.1428571429 0.125 0.1052631579 1 0.2089552239 0.3181818182 0.6086956522

FUNCTION * UNIQUEVERTX *

Given some face F0 of a Cepheid Ceph and a vector Vec
Programm determines the unique index I0 such that
coordinate I0 maximizes Vec.Normal F0. Thus,
F0 + Vec[I0].E^I0 is a candidate for a face of a Cepheid
with k+1 deguas.

Input:
CV is Ceph
Vec
That is, Vec has length as given by number of columns of
Ceph. Therefore syntax must be

* I0 + F0 Unique A0, [1] Vec *

Example:
BLUEF7 UNIQUEVERTX BLUE, [1] 1 1 1 1 1 1

FUNCTION * FACN2KARB *

For N=2 and k arbitrary ≥ 2, Program computes the face to
reference vector R=(1, ..., 1, 2, 1, ..., 1) of a Cepheid given by
A (K×2)-MATRIX A.

SYNTAX:
FACE + R ΔFACN2KARB A

BEISPIEL:
(Q BLUE)[;1 2]
110 55
130 33
150 77
170 99
190 11
210 1
230 49

1 1 2 1 1 1 ΔFACN2KARB (Q BLUE)[; 1 2]
1 1 1 0 1 1 1
0 0 2 2 0 0 0

FUNCTION * SUBEXTF *

SUBROUTINE FOR ....
GIVEN A FACE F OF DIMENSION (SAY) N × (K-1),
PROGRAM EXTENDS F INASMUCH AS AN ADDITIONAL
COLUMN IS INSERTED AS COLUMN K1. THIS COLUMN IS THE VÉKTÖR
(0, ..., 0, K1, 0, ..., 0) OF LENGTH K. HENCE THE RESULTING FACE
HAS DIMENSIONS N × K.

HYPOTHETICALLY, A*I0 IS THE MAXIMIZING VECTOR RESULTING
FROM PROGRAM * UNIQUEVERTX * WHILE K1 IS A COORDINATE
AT WHICH SOME REFERENCE VECTOR R SHOWS A 1.

K1I0 IS (K1, I0). HENCE SYNTAX SHOULD BE
* (K1, I0) SUBEXTF F *

EXAMPLE:
BLUEF7
Section 5: 3rd Algorithm

4.5 SUBEXTF BLUEF7 (HAENGTE DIE LETZTE SPALTE AN)

FUNCTION * ΔFCR *

SYNTAX:
FAM = ΔFCR A

GENERATES ALL FACES OF A CEPHOID REPRESENTED BY THE
K × N MATRIX A. BASED ON THE RECURSIVE ALGORITHM USING
THE MATRIX OF REFERENCE VECTORS.

OUTPUT IS A FAMILY OF FACES, I.E., A VECTOR EACH COMPONENT
OF WHICH IS AN INCLUDED N × K MATRIX REPRESENTING A FACE.

EXAMPLE:

\[
\begin{array}{cccc}
2 & 1 & 2 & 3 \\
2 & 1 & 2 & 3 \\
2 & 1 & 2 & 3 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
4 & 4 & 4 & 4 \\
4 & 4 & 4 & 4 \\
4 & 4 & 4 & 4 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
\end{array}
\]
FUNCTION * ΔFCRTIME *

SYNTAX:

FACES ← ΔFCRTIME A

THIS IS FUNCTION *ΔFCR*. HOWEVER, A TIME MEASURING IS INCLUDED. THE TOTAL TIME FOR TO COMPLETE THE COMPUTATION OF ALL FACES OF A CEPHOID IS RECORDED.

DATA AFTER EXECUTION APPEAR IN A GLOBAL VARIABLE * RECORD * THIS USES THE LITERAL VARIABLE (FORM) * RECFORM * AS PROVIDED IN THE WS.

EXAMPLE:

CEPHOID * LARBLUE * IS GIVEN VIA THE MATRIX

1 10 130 150 170 190 210 230
3 55 33 77 99 11 1 49
12 98 112 133 14 67 44 23
1 10 23 49 34 26 83 97
2 58 39 86 111 30 19 70
FACES ARE LISTEN IN VARIABLE ∗ LARBLFACES ∗. THE TIME ELAPSED TO COMPUTE THESE FACES (ABOUT 1 MIN 11.5 SECS) WAS STATED IN ∗ RECORD ∗ AS FOLLOWS:

RECORD

<table>
<thead>
<tr>
<th>Time elapsed (s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Begin time</td>
</tr>
<tr>
<td>2</td>
<td>End time</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
</tr>
<tr>
<td>6</td>
<td>h</td>
</tr>
<tr>
<td>7</td>
<td>m</td>
</tr>
<tr>
<td>8</td>
<td>s/1000</td>
</tr>
<tr>
<td>9</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>10</td>
<td>0 0 0 0 1 12</td>
</tr>
<tr>
<td>11</td>
<td>-676</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

For completeness we reproduce here the final result of this WS which is provided by the function ∗ ΔFCR ∗

[0] FACGLOB ← ΔFCR A;K;N;D;RVGLOB;K1;RVG11;A1;FAM1;REFM1;K2;F1;I0;F;R1
[1] ♦
[2] ♦ Generates all faces of a cepheid represented by the
[3] ♦ k × n matrix A. Based on the recursive algorithm using
[5] ♦
[6] ♦ Output is a family of faces, i.e., a vector each component
[7] ♦ of which is an included n × k matrix representing a face.
[8] ♦
[9] ♦
[10] ♦
[11] ♦ DATA:
[12] ♦ K ← (pA)[1]
[13] ♦ N ← (pA)[2]
[14] ♦ K = 2 or N=2: REDUCTION TO PROGRAMs ACCORDINGLY.
[15] ♦
[16] \( \rightarrow (K > 2) / \text{CONK} \)
[17] \( \text{FACGLOB} \leftarrow \Delta \text{FACK2NBEL} \text{ A} \)
[18] \( \rightarrow 0 \)
[19] \( \vdash \)
[20] \( \text{CONK:} \)
[21] \( \vdash \)
[22] \( \rightarrow (N > 2) / \text{CONN} \)
[23] \( \text{FACGLOB} \leftarrow \Delta \text{FACN2KBEL} \)
[24] \( \rightarrow 0 \)
[25] \( \vdash \)
[26] \( \text{CONN:} \)
[27] \( \vdash \text{IF N EXCEEDS K, THEN USE THE DUAL VERSION:} \)
[28] \( \rightarrow (N \leq K) / \text{PRIMAL} \)
[29] \( \text{FACGLOB} \leftarrow \text{DUALFAM} \Delta \text{FCR F A} \)
[30] \( \rightarrow 0 \)
[31] \( \vdash \)
[32] \( \text{PRIMAL:} \)
[33] \( \text{FACGLOB} \leftarrow 0.0 \)
[34] \( \vdash \text{THE MATRIX OF REFERENCE VECTORS (K, N):} \)
[35] \( \text{RVGLOB} \leftarrow K \text{ ALLREFVECS N} \)
[36] \( \vdash \)
[37] \( \vdash \text{BEGIN OF ROUTINE:} \)
[38] \( \vdash \text{INITIATE ROWS OF RVGLOB:} \)
[39] \( K1 = 0 \)
[40] \( \text{UP: } \rightarrow (\ K < K1 + K1 + 1) / 0 \)
[41] \( \vdash \text{ROWS OF RVGLOB WITH ENTRY 1 AT K1:} \)
[42] \( \vdash \)
[43] \( \text{RVGL11} \leftarrow (\text{RVGLOB}[:, K1] = 1) / [1] \text{ RVGLOB} \)
[44] \( \rightarrow (0 = (\rho \text{RVGL11}[1]) / \text{UP} \)
[45] \( \vdash \)
[46] \( \vdash \text{DETERMINE FACES OF THE CEPHOID OBTAINED BY A}^\Delta (\text{-K1}), \)
[47] \( \vdash \text{ITS FACES AND THE CORRESPONDING REFVECTORS.} \)
[48] \( \vdash \text{THIS CONTAINS INDUCTIONSTEP ...} \)
[49] \( A1 \leftarrow (1 - K1 = \tau K)[1]A \)
[50] \( \text{FAM1} \leftarrow \Delta \text{FCR A1} \)
[51] \( \text{REFM1} \leftarrow \text{REFVECMAT FAM1} \)
[52] \( \vdash \)
[53] \( \vdash \)
[54] \( \vdash \text{INITIATE ROWS OF RVGL11} \)
[55] \( \vdash \)
[56] \( K2 = 0 \)
[57]  UPP: \rightarrow (\rho_{RVGL11})[1] < K2 \rightarrow K2+1)/DELET
[58]  \text{REDUCE THE ROW OF RVGL11 AT HAND AT K1:}
[59]  R1 \leftarrow RVGL11[K2;]
[60]  R1 \leftarrow (1 - K1 = \tau K)/R1
[61]  \text{AND CONSIDER THE FACE GENERATED FOR THIS VECTOR}
[62]  \text{WRT THE REDUCED MATRIX A1:}
[63]  F1 \leftarrow FAM1 SEEKFACE RFSM1,[1] R1
[64]  \text{LOOK FOR THE OPTIMAL VERTEX OF A^(K1) RELATIVE TO F1:}
[65]  I0 \leftarrow F1 UNIQUEVERTX A1,[1]A[K1;]
[66]  \text{EXTEND F1 ABD ADD TO THE FAMILY IN CONSTRUCTION:}
[67]  F \leftarrow (K1,I0) SUbjectF F1
[68]  FACGLOB \leftarrow FACGLOB,[1] \subset F
[69]  \rightarrow UPP
[70]  \text{DELET:}
[71]  RVGLOB \leftarrow K1 CANCL RVGLOB
[72]  \rightarrow UP


