ED 067 251	SE 014 522
AUTHOR	Creveling, Cyrus J., Ed.
TITLE	Experimental Use of A Programming Language (APL) at the Goddard Space Flight Center.
INSTITUTION	National Aeronautics and Space Administration, Greenbelt, Md. Goddard Space Flight Center.
REPORT NO	GSFC-X-560-68-420
PUB DATE	Nov 68
NOTE	62p.
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Computers; *Computer Science; *Mathematical Applications; Programing; *Programing Languages
IDENTIFIERS	Goddard Space Flight Center

ABSTRACT

This document explains A Programming Language (APL) and describes the experiment that the Information Processing Division has undertaken to introduce APL to the Goddard Scientific Community. A brief historical sketch of steps taken to date is included and appendices giving illustrative examples of how APL actually has been used at the Goddard Space Flight Center are provided. (Author/DT)

ED 067251

X-560-68-420

NASA

SE

EXPERIMENTAL USE OF A PROGRAMMING LANGUAGE (APL) AT THE GODDARD SPACE FLIGHT CENTER

EDITED BY
CYRUS J. CREVELING

NOVEMBER 1968

GSFC

GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

X-560-68-420

EXPERIMENTAL USE OF A PROGRAMMING LANGUAGE (APL) AT THE GODDARD SPACE FLIGHT CENTER

Edited By

Cyrus J. Creveling
Information Processing Division

November 1968

Goddard Space Flight Center
Greenbelt, Maryland

EXPERIMENTAL USE OF A PROGRAMMING LANGUAGE (APL) AT THE GODDARD SPACE FLIGHT CENTER

Edited By

Cyrus J. Creveling
Information Processing Division

ABSTRACT

This document is intended to explain what APL is, and to describe the experiment that the Information Processing Division (IPD) has undertaken to introduce APL to the Goddard Scientific Community. We have prepared a brief historical sketch of steps taken to date and have provided some illustrative examples of how APL has actually been used at the Goddard Space Flight Center.

CONTENTS

EXPERIMENTAL USE OF A PROGRAMMING LANGUAGE (APL) AT THE GODDARD SPACE FLIGHT CENTER

WHAT APL IS

APL is an abbreviation of A Programming Language, a mathematical development of Dr. K. E. Iverson and associates having special attributes for the design and specifications of digital computing systems, both "hardware" and "software." It embraces ordinary computational arithmetic, algebraic formulae, the logical calculus (Boolean Algebra), and has special features for matrix manipulations. Although the language stands on its own as a mental concept, and as such is not implicitly related to any computational device (it is not "hardware oriented"), it has been "implemented" on more than one large scale computer and some smaller ones. This fact is of considerable importance in attempting to assess the future impact on the computer programming field, since APL is capable of competing with other computer languages including such well-established ones as Fortran and Algol.

SOME CHARACTERISTICS OF APL

Being a form of mathematical notation, APL has most of the attributes of the more common forms. It is composed of a small number of primitives, and these can be rigorously combined or redefined in terms of each other in a useful manner. Like algebra and trigonometry, this allows a continual refinement of statements, originally long and diffuse, into shorter and more elegant forms. This flexibility and the conciseness to which it leads is useful in that it makes possible the successive compaction of long but well-understood expressions into short recognizable terms.

This feature leads to Iverson's "aesthetic criteria" design attribute, wherein the designer (or program writer) is guided as much by his intuitive "feel" for the tractability of his problems as with a strictly rigorous and systematic development.

APL is easy to learn, because it has a simple syntax, relatively few primitives and new symbols (most of which have a mnemonic structure), and makes a maximum use of existing mathematics. These facts make it approach the condition of being self-documenting, since it is analogous to a mathematical derivation in which few marginal notes or parenthetical explanations are required.

APL is uncommitted to any particular technology or type of problem. Its character set (available on a "selectric" type ball) can be accommodated by a standard typewriter of approximately 100 characters (upper and lower case).

HISTORICAL BACKGROUND

APL was introduced to GSFC in 1965 by Dr. E. P. Stabler, University of Syracuse, a summer employee, in the performance of a computer design task. The example of his application of APL to a complex hardware/software design problem excited our interest in seeing if a widespread application of such a so-called Higher-Order Language (HOL) to typical Goddard problems might solve certain outstanding semantic problems endemic in our establishment. These problems of mutual understanding between the several disparate disciplines, forced into cooperative space projects of great complexity, are often referred to as "failure to communicate" or semantic difficulties. The results of these problems are felt in the long and difficult procedures necessary to eliminate malfunction, nonfunctions, and misfunctions, before and after satellites are launched. Under the press of tight schedules, these problems occasionally degenerate into personal recriminations, when a dispassionate review would probably indicate a mutual misunderstanding based on an incomplete or ambiguous specification at an interdisciplinary interface.

Dr. Stabler's thesis, taken from Dr. K. E. Iverson, author of A Programming Language (Wiley, 1962), is that APL, being basically a form of mathematics, and amenable to standard mathematical manipulation, is precise, comprehensive, and can be understood by physicists, engineers, and computer programmers alike. It embraces standard computational notation, Boolean operators, logical calculus, and has some novel features of particular power in matrix manipulations and in standard computer operation (such as sorting, listing, etc.).

Following these principles, the Information Processing Division and the Employee Development Branch (MUD) sponsored a short course in Higher-Order Languages. This course, taught at Goddard in April 1966 by Dr. Yaohan Chu of the University of Maryland, explored the applicability of several computer languages to digital design and computational problems.

Since the results of the course were generally favorable, we organized a Higher-Order Language Seminar, held at GSFC June 16, 1966, under the chairmanship of Dr. George H. Ludwig. Panelists were Dr. K. E. Iverson, IBM; Dr. C. A. Wogrin, Yale University; Dr. E. P. Stabler, Syracuse University; Dr. Yaohan Chu of the University of Maryland; Mr. Thomas Gorman and Mr. Cyrus J. Creveling of GSFC (all technologists): and Mr. James Bostain of the Institute of Foreign Affairs, a linguist. The merits of various approaches to the computer design and specification and programming language problems were discussed and from these discussions came the resolve on the part of the IPD to conduct an in-depth experiment in the use of APL. The deciding factor was the information that APL had been implemented on an IBM System 360 computer with many remote terminals connected in a time-sharing mode. With an offer from Dr. Iverson to conduct a class in APL at Goddard, using 10 IBM-type 1050 computer terminals, we planned a class of between 30 and 40 Goddard personnel for a two-week course. First, a select group consisting of Dr. George Ludwig, Mrs. Melba Mouton, Mr. Thomas P. Gorman, Mr. Bill Mish, Mr. William Alford, Mr. Larry Hyatt, and Mr. Cyrus J. Creveling from GSFC spent a week at the Watson Research Labs, indoctrinating Dr. Iverson's staff in the types and range of problems of interest to Goddard. The ensuing 10-day course was well attended and created very favorable impressions. A number of demonstrations of APL were made during these two weeks, including one to Goddard's Director, Dr. Clark, and to Mr. Mengel, Assistant Director for Tracking and Data Systems.

Following the initial course at GSFC, three IBM 1050 computer terminals were retained in the Information Processing Division to provide continuity of use, and these have been augmented since then so there are now six APL terminals connected by leased telephone lines to Yorktown Heights, New York. (For this use we pay nominal fees for connection to the computer of \$2.50/hour, and for central processor time of \$2.20/minute, per terminal.)

As a means of attracting attention to our APL experiment, the program committee of the Engineering Lecture Series was induced to invite Dr. K. E. Iverson of the IBM Watson Research Center, Yorktown Heights, New York, to make an address in the Goddard Engineering Lecture Series on January 18, 1967. This lecture featuring the use of a working computer terminal on the stage of the auditorium and a closed-television view of the keyboard on the screen attracted a "standing room only" audience.

In the ensuing years (1967, 1968), the Information Processing Division (IPD) in conjunction with GSFC Employee Development Branch, Manpower Utilization Division, has offered a succession of short courses in APL in which about 220 individuals have been afforded an opportunity to learn enough of the language to make use of it in their everyday work. Staff members of the IPD taught a short course for the benefit of the 1967 Summer Workshop, one of the results of which is published in the article by Dr. Mansour Javid.

CHRONOLOGY OF MAJOR EVENTS

Yaohan Chu, An Introduction to Higher-Order Languages, GSFC, 1966
Higher-Order Language Seminar, June 16, 1966
Preparatory Class, Yorktown Heights, October 10-14, 1966
Iverson's Class in APL, October 1966
Morton's Class in APL, March 1967
Short Course, June 19-27, Alford, Bonk, Fleming, McNamara, Vaughan, Bouricious, Instructors
Higher-Order Language Seminar, November 22, 1967, Bob Fisk's Class in APL
Iverson's Lecture (Engineering Lecture Series) (Tape)
Applied Mathematician's Video Experiment at Yorktown Heights, June 1968
Pilot Video Course in APL at Goddard Space Flight Center, 1968

EXPERIENCES OF SELECTED INDIVIDUAL USERS

There have been numerous individuals who have had their curiosity excited to the extent of learning APL on their own, and this is entirely feasible with or without a computer terminal available. When polled, members of the first class, which made extensive use of the terminals, were about evenly divided as to the necessity of having the terminal as an adjunct to the class.

A short-term visitor to Goddard from Centre National D'Etudes Spatiales (CNES), Yves Leborgne described his learning procedure, which consisted of studying pages 200-201 of the IBM System Journal (Vol. 3, Nos. 2 & 3, 1964), in an article by Falkoff, Iverson, and Sussenguth. A brief list is given of the operations, functions, and relations with illustrative examples he worked in a very few hours, reworked in less than one hour, repeated the third time in a few minutes, and then began writing programs on his own.

Dr. Mansour Javid of the City University of New York and GSFC, was the principal investigator of the 1966 Summer Workshops in the information area. As such, he was charged with organizing the program of a group graduates and facility members, and supervising their work, normally a full-time job. He further occupied himself with carrying a personal investigation into a computer programming problem which had been estimated to cost about $25,000 if performed by contract. He finished this job and wrote the report as Goddard document X-200-67-639. That report does not explicitly acknowledge the underlying reliance on APL, but Dr. Javid's account here makes it clear.

The named report describes the Fortran programs appended hereto, and the main sections occupy 50 pages.

More typically, a visitor can sit down at one of our IBM Type 1050 computer terminals which usually have an APL manual handy, and by reading the instruction on how to "turn on," can then follow the explanation of how the language operates in order to work the examples in the manual. A few hours of this allows anyone with a mathematical background the use of the terminal for solving complex computational, logical, and manipulative problems. Actually, a grade school student can learn to use the system in the desk calculator mode in a few minutes. Undergraduate summer trainees have accomplished significant achievements.

CONCLUSIONS

The results of these experiments in using APL have born out our expectations that APL is well suited for designing or describing the various components of an information system: The data entering the system; the special-purpose and general-purpose machines (computers) which process the data; and the mathematical formulations and programs used to direct the manipulations and computations performed on the data. Each of these functions have been performed by several investigators and are well authenticated and documented. This use of APL was the one which originally interested us.

The experiences of many individuals at GSFC have indicated that APL is a promising development in computer language, which in the most enthusiastic vein may replace many of the widely-used general-purpose computer languages which are oriented to mathematical and logical problems, e.g., Fortran, Algol, and MAD. This is by no means unanimous, since the least enthusiastic concede little usefulness to it, and when one group of 25 or more were polled concerning its suitability to a number of applications, there were wide divergences of opinion on all categories. The concensus, however, remains favorable. As in all endeavors, there are a few individuals whose enthusiasm is unbridled and their views must be discounted, as much those of the few individuals who are antagonistic to most new ideas.

For APL to attain wide usage at GSFC, a large number of terminals such as IBM 1050's or IBM 2741's must be distributed so that they can be easily reached by anyone. Two to five terminals on each floor of each major building would probably satisfy most general needs. Some heavy users might require exclusive use in their office; in some areas rooms containing several terminals would be necessary. This number of terminals would require a computer the size of a System 360/50 which could support about 100 terminals, or approximately 1000 users. (Use of the Disc Operating Systems (DOS), an IBM executive program for medium-sized System/360 machines, will allow background work to be done simultaneously.) In the near future, it is expected that a version of APL operating under Operating System (O. S.), the large computer executive used at Goddard, will be available. This will make it possible to incorporate APL into our large computers.

There is presently a deficiency to APL as a computational aid: Most input and output is by keyboard, which rules out operation on large data bases with fast input. It appears that a mode whereby a program could be compiled to be run in a batch process on some other machine would be desirable.

Since APL operates on a conversational mode, each statement is interpreted and debugged as it is written, and a large percentage of the inadvertent programming errors can be eliminated while the program is being written. This feature could have a major impact on conventional program writing, since program debugging is presently using 20 to 30 percent of the time of our big machines.

APL, being mathematical by nature, is easy for scientists and engineers to learn to use, and as such allows them direct access to the power of a computer without the professional programmer as a middle-man. This can be a great benefit to the creative individual, since the delays of conventional programming and computer "turn around" time can discourage and stultify the creative process.

The use of APL as a design and specification language for system (both hardware and software) provides a concise and precise means of communication between the various disciplines involved in designing and operating an information system. These usages have been demonstrated at GSFC most forcefully by Dr. E. P. Stabler, and D. H. Schaefer (see Appendix B), and these descriptions have been used as the basis for building a working computer and its software by W. Webb, and as a specification for a procurement request by Schaefer. The long-term benefits of the use of APL in this context are hard to assess, but experience with difficulties in the past lead us to believe they will be profound; indeed, the Information Processing Division became convinced of the necessity of a language like APL before it was known that a machine implementation existed. IPD still believes that this is a sufficient reason to pursue the use of APL, even though this use alone would directly affect only a few people and would not require extensive facilities.

Although many of the aims of our experimental use of APL have been met, we can say that the current usage made of our facilities justifies their continued existence. A considerable expansion could be justified on an operational need and in accordance to other reasons cited above. It is unlikely that a demand for expansion will rise spontaneously, since many people who would use facilities if they existed "don't know what they are missing," and are unlikely to be enticed into trying them on the rather vague supposition that if they find it useful, that the demand will be met by "someone" filling it. Rather, our cautious but progressive exploration of the many initially unknown factors have indicated that there is a potentially great benefit to Goddard scientists, but that it has now reached a place where positive, purposeful management must assess the situation and provide these facilities in order that the benefits be attained. This involves a degree of risk which our investigations have led us to believe are negligibly small.

Another factor over which GSFC does not have complete control (but does have some influence) is the present reluctance of IBM to provide APL to the public as "product line." There is an understandable reluctance on the part of manufacturers to undertake the extensive support of yet another computer language when their resources to deliver the software to which they are already committed are fully occupied.

APL has developed over a period of several years as a research effort and has only recently arrived at the stage where it is ready to be exploited on a larger scale. This venture is being urged mostly by people who have become aware of APL from the description of the language in the literature, through use of APL at the Watson Research Center where the development was carried on, and at the few organizations at which experimental usage has been tried.

ACKNOWLEDGMENT

The editor acknowledges the assistance accorded by all of the individual contributors to this report, since it is their experience which provides the substance upon which all conclusions as to the value of APL and our experiment stands: The International Business Machines Corporation through the administration of Dr. K. E. Iverson and his staff at the Watson Research Lab, Yorktown Heights, and several individuals in their Washington Office, notably, Arthur Tonkinson, Mal Morton, Robert Fisk, and Dick Detmering, who have supplied invaluable assistance in providing instructors, facilities, and in the first few courses, a number of individual speakers who illustrated their experience in APL.

The most important contribution has been the initiation of the first few stages of this experiment by Dr. E. P. Stabler, and the enthusiastic advice and support of Dr. C. A. Wogrin, both consultants of the Information Processing Division.

BIBLIOGRAPHY

1. Iverson, K. E.: A Programming Language, Wiley, 1962.

2. Iverson, K. E., Falkoff, and Sussenguth: A Formal Description of the System 360, IBM Journal.

3. Iverson, K. E.: Elementary Functions, S.R.A.

4. Stabler, E. P.: Telemetry Computer Studies, GSFC Summer Workshop, 1965, GSFC document X-100-65-407.

5. Sobotkin, F.: Use of a Formal Language to Describe On-Board and Ground Station Processing of Satellite Experiment, GSFC Summer Workshop, 1966, GSFC document X-700-67-94.

6. Creveling, C. J.: A Unified Information Processing System, Adaptive Telemetry Conference, GSFC document X-731-67-7, pp. 135-140, January 1967.

7. Stabler, E. P., and Creveling, C. J.: Spacecraft Computers for Scientific Information Systems, Proc. IEE, vol. 54, no. 12, December 1966.

8. Cole, Isabella J., and Steinberg, Ellen L.: Some Applications of a Programming Language, GSFC document X-522-67-367, August 1967.

9. Schaefer, D. H., and Snively, J. W., Jr.: The Plasma Statistics Computer Aboard Explorer XXXIV, GSFC document X-711-67-520, October 1967.

10. Javid, Mansour: A Digital Computer Program for Calculating the Radiation Pattern of an Antenna of Arbitrary Geometry With Arbitrary Primary Feed, GSFC document X-200-67-639.

Appendix A

THE USE OF APL IN AN ANTENNA DESIGN PROBLEM

Dr. M. Javid*

During the Goddard Summer Workshop of 1967, I was assigned the task of writing a computer program for calculating the radiation pattern of a reflector of arbitrarily defined shape, illuminated by a primary source with arbitrarily defined characteristics and location. The geometry and source characteristics were to be handled by subroutines, thus making it possible to use a main driving program and different subroutines corresponding to the different geometries and specifications. It was understood that a subroutine corresponding to a given geometry would differ only in a few instructions from a subroutine for another geometry, so that the engineers could easily make the modifications needed to obtain the new subroutine from a previously used routine.

The required flexibility of program indicated a great detail of experimentation with different structures for the main program and the subroutines. Since the writer is an amateur programmer, and at the time was only acquainted with Fortran, it seemed that the task could not be completed during the period of the Summer Workshop.

Fortunately, at this time classes were being held to acquaint the Goddard staff with APL. After attending three one-hour periods of these classes, the writer realized that, with the availability of APL terminals, it would be possible to experiment with various structures for the program, to test the programs, and to amend them in a very short time. In fact, within two weeks after attending the three hours of instruction, many combinations of main programs and subroutines were tested. It was soon clear that the original structure conceived by the writer would not have the required flexibility, and the final program had very little in common with the original concept. However, the drastic changes, which would have required many months of time (waiting for return of the results from a batch operated computer center), consumed less than two weeks, pleasantly spent at the APL terminal at the writer's convenience. The attached program was the final version of the work done in APL. It was then translated in Fortran and greatly expanded to provide for input-output handling of data and results.

* City University of New York, GSFC Summer Workshop Principal Investigator. See GSFC Document X-200-67-639, same title, by M. Javid.

The reader familiar with APL will not fail to observe that the program is written by a novice who learned enough to do a job. This is one of the important features of APL. It is possible to learn enough to write a useful program without having to know all there is in the language.

Even with the cursory acquaintance with the language, I could see its power, and in particular, its advantages in dealing with arrays and matrices.

Whenever the Fortran program based on the APL model runs into trouble, the debugging of the trouble was facilitated by running tests on the APL program. This again resulted in a great saving in waiting time, minimizing the debugging time needed at the batch-run computer.

```
      )FNS
BEAM    COS      DS       HXI      HXR      HYI      HYR      HZI      HZR      HXZ

      ∇BEAM[⎕]∇
    ∇ RES←AR1 BEAM AR2
[1]   DIA←30
[2]   LMDA←0.425
[3]   DFID←3
[4]   DTAD←3
[5]   TPI←2×3.141592654
[6]   DR←TPI÷360
[7]   DFI←DFID×DR
[8]   DTA←DTAD×DR
[9]   K←TPI÷LMDA
[10]  DRHO←S←LMDA÷4.731666667
[11]  R←⌊(DIA÷2)÷S
[12]  M←999
[13]  RI←(R+2)ρ0
[14]  NSUM←(R+2)ρ0
[15]  J←0
[16]  I←0
[17]  B13:I←I+1
[18]  SUM←0
[19]  B12:J←J+1
[20]  →(J>R)ρB10
[21]  DSUM←6×J
[22]  →(DSUM>M)ρB14
[23]  SUM←SUM+DSUM
[24]  →(SUM>M)ρB11
[25]  →B12
[26]  B11:RI[I+1]←J-1
[27]  NSUM[I+1]←SUM-DSUM
[28]  J←J-1
[29]  →B13
[30]  B14:'STORAGE IS INSUFFICIENT FOR THE FOLLOWING RING:'
[31]  J
[32]  RI[I+1]←J-1
[33]  NSUM[I+1]←SUM
[34]  I←I+1
[35]  →B15
[36]  B10:'END OF REFLECTOR SPECIFICATION'
[37]  RI[I+1]←J-1
[38]  NSUM[I+1]←SUM
[39]  I←I+1
[40]  B15:LIR←I
[41]  'TOTAL NUMBER OF ELEMENTARY AREA CONTRIBUTING TO THIS COMPUTATION IS:'
[42]  +SUMDS←+/NSUM
[43]  MTX←(9,((AR1+1)×AR2+1),LIR-1)ρ0
[44]  VX←Mρ1
[45]  VY←Mρ1
[46]  VZ←Mρ1
[47]  VNX←Mρ1
[48]  VNY←Mρ1
[49]  VNZ←Mρ1
[50]  VDS←Mρ1
[51]  VHXR←Mρ1
[52]  VHXI←Mρ1
[53]  VHYR←Mρ1
[54]  VHYI←Mρ1
[55]  VHZR←Mρ1
[56]  VHZI←Mρ1
[57]  IR←0
[58]  B16:IR←IR+1
[59]  →(IR=LIR)ρ0
```

```
[61]  T←NSUM[IR+1]
[62]  IEND←0
[63]  I←1
[64]  J←-1
[65]  B1:J←J+1
[66]  →(J=(RT[IR+1]-RT[IR]))ρB3
[67]  JH←RT[IR]+J+0.5
[68]  RHO←C×JH
[69]  I←I-1
[70]  IENDO←IEND
[71]  NUM←6×JH+0.5
[72]  DPHI←TPI÷NUM
[73]  IEND←IENDO+NUM
[74]  B2:I←I+1
[75]  →(I=IEND+1)ρB1
[76]  IH←(I-IENDO)-0.5
[77]  PHI←DPHI×IH
[78]  VX[I]←X←RHO×COS PHI
[79]  VY[I]←Y←RHO×SIN PHI
[80]  VZ[I]←Z←X ZI Y
[81]  VNX[I]←X NXZ Y
[82]  VNY[I]←X NYZ Y
[83]  VDS[I]←X DS Y
[84]  VHXR[I]←X HXR Y
[85]  VHXI[I]←X HXI Y
[86]  VHYR[I]←X HYR Y
[87]  VHYI[I]←X HYI Y
[88]  VHZR[I]←X HZR Y
[89]  VHZI[I]←X HZI Y
[90]  →B2
[91]  B3:NI←-1
[92]  B5:NI←NI+1
[93]  →(NI=AN1+1)ρB16
[94]  NJ←-1
[95]  B9:NJ←NJ+1
[96]  →(NJ=AN2+1)ρB5
[97]  FPNO←FPNO+1
[98]  IXR←IXI←IYR←IYI←IZR←IZI←0
[99]  TA←NI×DTA
[100] FI←NJ×DFI
[101] STA←SIN TA
[102] CTA←COS TA
[103] CFI←COS FI
[104] SFI←SIN FI
[105] I←0
[106] B7:I←I+1
[107] →(I=T+1)ρB6
[108] KD←K×((VX[I]×CFI×STA)+VY[I]×SFI×STA)+VZ[I]×CTA
[109] CKD←COS KD
[110] SKD←SIN KD
[111] AXR←(VNY[I]×VHZR[I])-VHYR[I]
[112] AXI←(VNY[I]×VHZI[I])-VHYI[I]
[113] IXR←IXR+((AXR×CKD)-AXI×SKD)×VDS[I]
[114] IXI←IXI+((AXR×SKD)+AXI×CKD)×VDS[I]
[115] AYR←VHXR[I]-VNX[I]×VHZR[I]
[116] AYI←VHXI[I]-VNX[I]×VHZI[I]
[117] IYR←IYR+((AYR×CKD)-AYI×SKD)×VDS[I]
[118] IYI←IYI+((AYR×SKD)+AYI×CKD)×VDS[I]
[119] AZR←(VNX[I]×VHYR[I])-VNY[I]×VHXR[I]
[120] AZI←(VNX[I]×VHYI[I])-VNY[I]×VHXI[I]
[121] IZR←IZR+((AZR×CKD)-AZI×SKD)×VDS[I]
[122] IZI←IZI+((AZR×SKD)+AZI×CKD)×VDS[I]
[123] →B7
[124] B6:CRR←(STA×CFI×IXR)+(STA×SFI×IYR)+CTA×IZR
[125] CTR←(CTA×CFI×IXR)+(CTA×CFI×IYR)-STA×IZR
```

```
[127] CRI←(STA×CFI×IXI)+(STA×SFI×IYI)+CTA×IZI
[128] CTI←(CTA×CFI×IXI)+(CTA×CFI×IYI)-STA×IZI
[129] CFI←(-SFI×IXI)+CFI×IYI
[130] AR2V←(CRR*2)+CRI*2
[131] AT2V←(CTR*2)+CTI*2
[132] AF2V←(CFR*2)+CFI*2
[133] MTX[;FRHO;IR]←CRR,CTR,CFR,CRI,CTI,CFI,AR2V,AT2V,AF2V
[134] AT2V
[135] →R9
[136] R4:→R16
      ∇

       ∇DS[⎕]∇
      ∇ DSI←X DS Y
[1]    DSI←TPI×(((RHO+0.5×S)*2)-(RHO-0.5×S)*2)÷2×NUM
      ∇

       ∇HXI[⎕]∇
      ∇ HXII←X HXI Y
[1]    HXII←0
      ∇

       ∇HXR[⎕]∇
      ∇ HXRI←X HXR Y
[1]    HXRI←0
      ∇

       ∇HYI[⎕]∇
      ∇ HYII←X HYI Y
[1]    HYII←ZR2×SKR
      ∇

       ∇HYR[⎕]∇
      ∇ HYRI←X HYR Y
[1]    R2←(X*2)+(Y*2)+Z*2
[2]    R1←R2*0.5
[3]    KR←K×R1
[4]    SKR←SIN KR
[5]    CKR←COS KR
[6]    ZR2←Z÷R2
[7]    HYRI←-ZR2×CKR
      ∇

       ∇HZI[⎕]∇
      ∇ HZII←X HZI Y
[1]    HZII←-YR2×SKR
      ∇

       ∇HZR[⎕]∇
      ∇ HZRI←X HZR Y
[1]    YR2←Y÷R2
[2]    HZRI←YR2×CKR
      ∇

       ∇HXZ[⎕]∇
      ∇ VHXZ←X HXZ Y
[1]    R1←((X*2)+(Y*2)+Z*2)*0.5
[2]    VHXZ←X÷(Z-R1)
      ∇

       ∇HYZ[⎕]∇
      ∇ VHYZ←X HYZ Y
[1]    R1←((X*2)+(Y*2)+Z*2)*0.5
[2]    VHYZ←Y÷(Z-R1)
      ∇
      ∇ZZZ[⎕]∇
    ∇ VRZZ←X ZZZ Y
[1]  • VRZZ←-1
    ∇

      ∇ZI[⎕]∇
    ∇ Z←X ZI Y
[1]    F←36
[2]    RHO2←(X*2)+Y*2
[3]    Z←(RHO2:(4×F))-F
    ∇

      ∇SIN[⎕]∇
    ∇ RR←SIN X;Z;S;R;Q0;S̲;C̲;Z2
[1]    S←,X<0
[2]    Z←,|X
[3]    R←0.7853981633974483|Z
[4]    Q0←8|Q0-1|Q0←Z:0.7853981633974483
[5]    S←S≠Q0>3
[6]    Q0←4|Q0+1
[7]    R←((R×2|Q0)+(~2|Q0)×0.7853981633974483-R):0.7853981633974483
[8]    RR←(ρZ)ρ0
[9]    →(0=+/Q0<2)/11
[10]   RR←(Q0<2)\(0.785398163397448+Z2×¯0.08074551218828051+Z2×0.002490394570188843+Z2×¯3.657620415891387E¯5+Z2×
       3.133616225433375E¯7+Z2×¯1.757150074693567E¯9+6.877360570940357E¯12×Z2←Z*2)×Z←(Q0<2)/R
[11]   →(0=+/Q0>1)/13
[12]   RR←RR+(Q0>1)\1+Z2×¯0.3084251375340423+Z2×0.01585434424381542+Z2×¯0.0003259918869267372+Z2×
       3.590860446027951E¯6+Z2×¯2.461136403365227E¯8+Z2×1.150051202818624E¯10+¯3.856189006132304E¯13×Z2←((Q0>1)/R)*
       2
[13]   RR←(ρX)ρRR×(1 ¯1)[S+ι1]
    ∇

      ∇COS[⎕]∇
    ∇ RR←COS X
[1]    RR←SIN 1.570796326794897-X
    ∇
```

Appendix B

APL SIMULATION OF THE STATISTICS COMPUTER ABOARD EXPLORER XXXIV

David H. Schaefer*

INTRODUCTION

Simulation of a computer by a computer can be most advantageous. An APL simulation of the special purpose computer known as the "statistics computer," aboard the earth satellite Explorer XXXIV, has proved valuable in at least three respects. First, it has provided a tool for troubleshooting the processor in the laboratory. For a given input, a description of the output and the state of various registers and counters is immediately available to be compared with these states in the device under test. Second, the APL description of the device has provided a unique form of documentation of exactly what the device is, that is, that it has provided a ready method of telling others exactly what the input-output characteristics of the device are. Third, the APL simulation of the device has provided an easy method of determining if specific given inputs can produce specific outputs that have been received from the orbiting spacecraft.

DESCRIPTION OF ON BOARD COMPUTER

The plasma experiment on the Explorer XXXIV satellite provided fertile ground for on-board data processing. In this experiment, a large amount of data is being produced by the sensing element, too much data to be completely transmitted over the available telemetry. In order to fit the experiment's data into the telemetry system, a parameter extraction technique was evolved where parameters related to mean, variance, and mode are computed aboard the satellite.

The purpose of the statistics computer is to efficiently represent the prime characteristics of data collected by the plasma detector during a full rotation about its axis of the spin stabilized spacecraft. Figure B1 shows the type of output expected from the plasma detector in the interplanetary space, and in the region of space between the magnetosphere and the solar shock wave.

*GSFC Flight Data Systems Branch

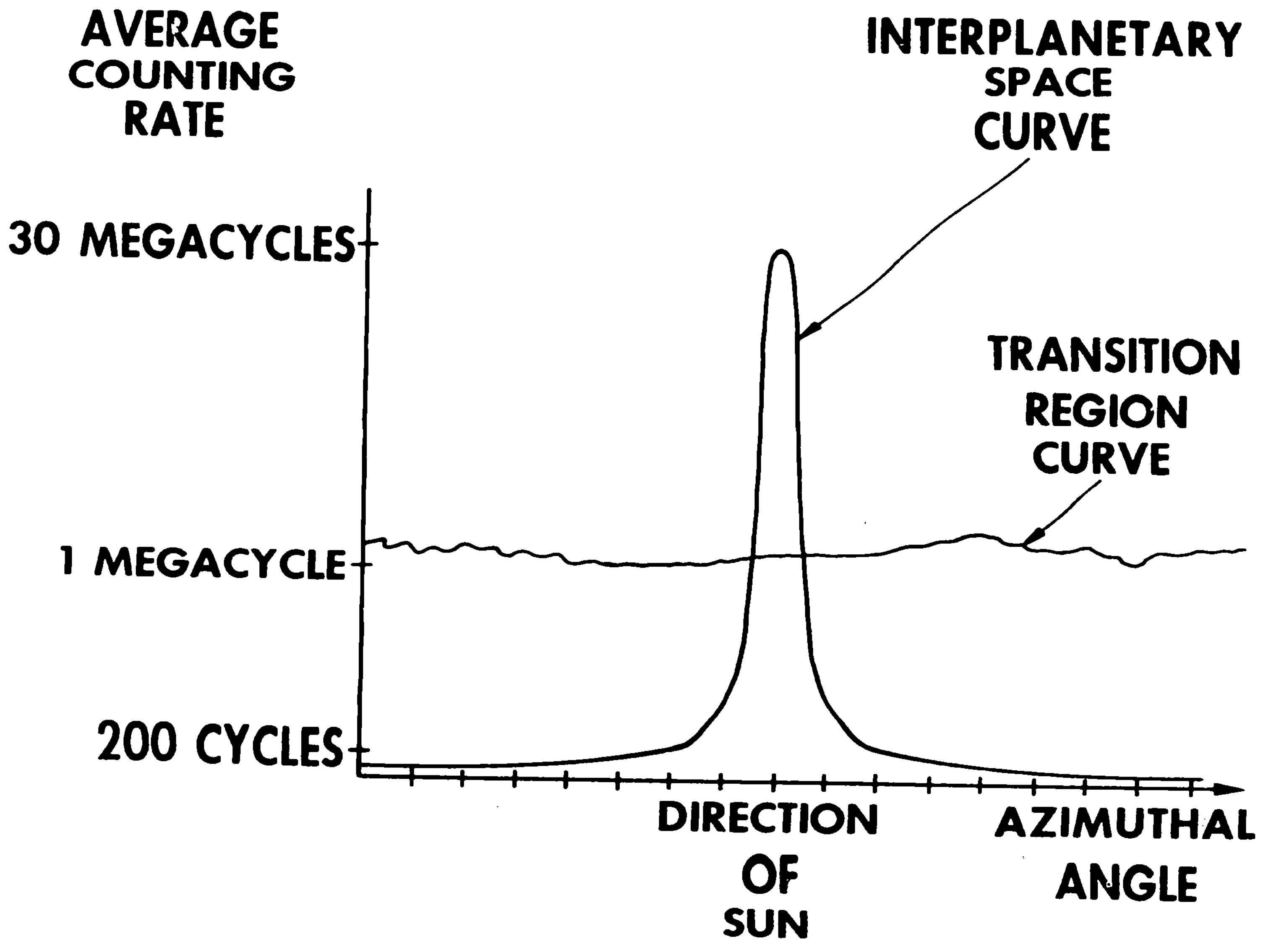


Figure B1—Typical input data.

In interplanetary space all plasma is expected to arrive from the direction of the sun, while in the transition region the plasma is expected to be homogeneously distributed. Questions of special interest are "Where is the boundary between the two regions?" and "How does the shape of an azimuthal angle versus counting rate curve change as the boundary is traversed?" In addition there are the usual questions of: "How much flux is present?" and "Where in each revolution was the counting rate the greatest?"

The process used to provide this information using a minimum number of bits is to calculate statistical quantities. Each revolution of the spacecraft about its spin axis is divided into 16 parts, each covering 22.5 degrees of the revolution. Defining C as a 16-element vector where C[I] is the number of pulses produced by the sensor in the ith sixteenth of the revolution, we can define a ratio r by the function RA, such that

```
      ∇ R←RA C
[1]     R←((+/(C*2))×16)÷((+/C)*2)
      ∇
```

This quantity can only assume values between one and sixteen. If r has a value of 16, then all inputs arrived in one-sixteenth of the revolution, and the input curve is close to the interplanetary space input of Figure B1.

On the other hand, if r is 1, equal inputs arrived during every sixteenth of the revolution such as the transition region curve of Figure B1. Ratios in between 1 and 16 are indicative of various well-defined in-between cases. On board the satellite only a rough approximation to r is calculated. To obtain this rough value, the difficult operations of multiplication and division are not performed. Instead, the sum-of-squares calculation is accomplished by a counting method. Four bits of this calculation are telemetered, these bits being determined by the logarithmic representation of the total number of counts received in the revolution. The logarithmic counter representation itself is also telemetered. Figure B2 shows this in block diagram form. From these quantities more refined values of r can be obtained on the ground.

It is possible to receive 2^{19} counts in any sixteenth of a rotation. To directly transmit the number of counts received in each sixteenth of a rotation over a complete rotation of the rotation of the spacecraft would therefore require 16×19 or 304 bits. By doing the process described here, a description of the information collected during a complete rotation of the spacecraft is represented by 16 bits, a bit saving of a factor of 19 over direct transmission of the data.

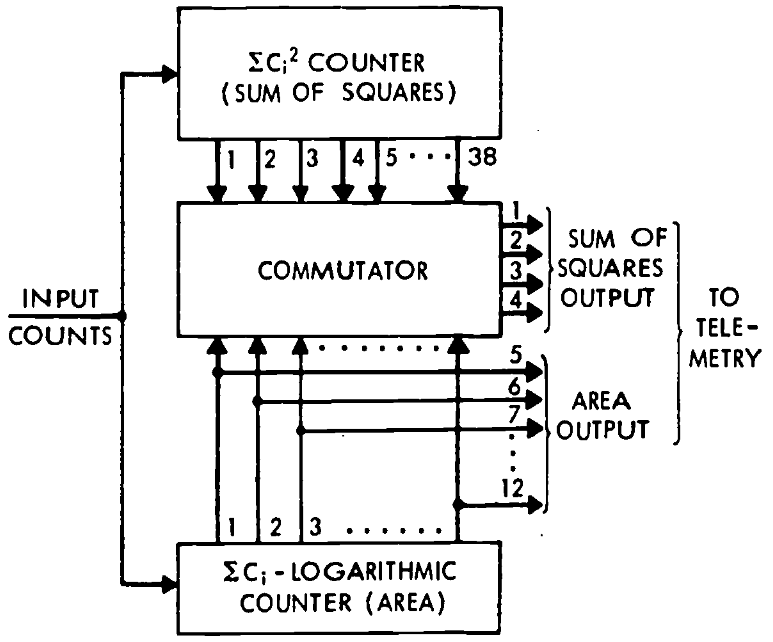


Figure B2— Block diagram of processor.

Figure B3 shows a segment of the output of a computer program that relates the output of the IMP flight hardware to the input data which produced the specified output. For example, if the logarithmic representation of the total number of pulses detected by the sensor (the A bits) is 195, the total number of counts actually detected lie between 38,912 and 40,959 counts. If, furthermore, the four transmitted bits of the squarer counter (the S bits) have, for instance, the value 5, the ratio of the input histogram must be between 6.39 and 8.52. From this it can be calculated that the largest bar of this input histogram is between 17,048 and 29,754 counts. For each of these quantities the harmonic mean (H.M.) of the range and the maximum ± percentage error (P.E.) are also listed.

In addition to the above mentioned computations, the statistics computer also provides an indication of which sixteenth of the revolution the number of received pulses was greatest. A detailed description of the computer can be found in the bibliography for this Appendix.

APL SIMULATION

Referring to Figure B2, the three blocks shown have been individually simulated on the APL system and then combined to produce the action of the on-board computer as a whole.

The logarithmic counter conversion of the number N is the following:

```
      ∇ L←CL N
[1]     MN←(19⍴2)⊤N
[2]     Z←(MN⍳1)⌊16
[3]     K←(4⍴2)⊤(16-Z)
[4]     J←Z⌊15
[5]     L←K,MN[(J+1),(J+2),(J+3),(J+4)]
      ∇
```

In practice N is +/C where C is the 16 component input vector.

The output (line 5) represents the binary data on lines 5 to 12 on the right side of Figure B2.

The squaring counter is a combination of two counters, values of one counter being transferred to another counter. Also involved is a prescaler of five stages. The following function takes into account all complications of prescaling, transfer pulses that follow an input pulse by 16 counts, and the fact that the commutator of Figure B2 will sense a "one" if it tries to commutate an open circuit. For an input C, the 27 actual lines (rather than 38 shown in Figure B2) coming from the sum-of-squares counter have impressed on them the binary number represented by the first part of the vector S in the following function:

```
       ∇SQ[⎕]∇
     ∇ S←SQ C
[1]    SW←⌊((SC C)÷32)
[2]    N←⍳15
[3]    TP←SW[1],SW[N+1]-SW[N]
[4]    CV←(32|(SC C))≥16
[5]    I←TP-(0,CV[⍳15])
[6]    S←((27⍴2)⊤(+/(I×(I+1))÷2)), 1 1 1
[7]    →0
     ∇
```

The prime element of the commutator of Figure B2 is a shift register containing only a single "one." This "one" is shifted by the logarithmic counter. For the input vector C, the position of the "one" (the LAtch position) is

```
           ∇LA[⎕]∇
         ∇  B←LA C
   [1]      B←(1+2⊥((8α5)/(CL+/C)))⌈6
         ∇
```

where the initial position is assigned the value six.

The value of the four commutated bits of the squaring counter (the "S Bits," lines 1, 2, 3, and 4 on right hand side of Figure B2) are a function of the logarithmic counter (via the function LA) and the squarer (via the function SQ). These bits are defined as

```
         ∇  G←SB C
   [1]      B←LA C
   [2]      S←SQ C
   [3]      G←S[(33-B),(34-B),(35-B),(36-B)]
         ∇
```

In addition to the boxes shown in Figure B2, the statistics computer also has circuitry to determine, and provide for telemetry, the location of the sixteenth of a revolution in which the largest number of counts were received. Taking into account several complicating factors including five stage prescaling, the value given as the sector of maximum input counts for the input vector C is given by the function

```
         ∇  A←MAX C
   [1]      SV←⌊(((SC C)+16)÷32)
   [2]      N←⍳15
   [3]      DV←SV[1],SV[N+1]-SV[N]
   [4]      A←16|(⌈/(((⌈/DV)=DV)/⍳16))
         ∇
```

The function SC above is the sum scan function that has not at the time of writing been implemented on the APL system. The following function for sum scan has, therefore, been derived for vectors of dimension 16:

```
      ∇ SS←SC C
[1]     CS←C[1],+/C[1 2]
[2]     N←3
[3]     CS←CS,+/(CS[N-1],C[N])
[4]     N←N+1
[5]     →((N≠17)×3)+((N=17)×6)
[6]     SS←CS
      ∇
```

The actual telemetry format for data collected during one revolution of the satellite consists of four sixteen-level pulse-frequency modulation bursts. The hexadecimal representation of this (the form of "quick look" data from the satellite) is given by the TELemetry function:

```
    ∇ P←TEL C
[1]   P←(2⊥((8α4)/(CL+/C))),(2⊥((8ω4)/(CL+/C))),(2⊥SB C),MAX C
    ∇
```

All the foregoing specify the computer and its telemetered outputs.

A simple function useful in helping to reduce received data is the LG function. This function reduces the first two components of the telemetry vector to the approximate number of input counts. Its definition is

```
      ∇ A←LG Y
[1]     →((Y[1]>1)×2)+((Y[1]≤1)×4)
[2]     A←⊥(1,((4ρ2)⊤Y[2]),1,((Y[1]-2)ρ0))
[3]     →0
[4]     A←16⊥Y
      ∇
```

Examples of the use of the TEL and LG functions follow. In the first three examples vector H has the ratio 16, vector E has the ratio 8, and the vector (16 ρ 2440) has the ratio 1. Figure B3 can be used in checking the answers for these three examples:

```
      H
0   0   0   0   0   39040   0   0   0   0   0   0   0   0   0   0
      TEL H
12   3   11   6
      LG 12,3
39936
      E
0   0   0   0   0   19500   19500   0   0   0   0   0   0   0   0   0
      TEL E
12   3   5   7


      TEL 16ρ2440
12   3   0   14


      A
1   2   300   400   500   600   70   800   90000   10   20000   12   13   14   15   16
      TEL A
13   11   7   9
      LG 13,11
112640
      +/A
112753


      TEL 2,15ρ3
2   7   7   6
      LG 2,7
47
      TEL 16ρ0
0   0   7   0
```

```
 ABITS                            AREA..    MIN.        MAX.        H.M.       P.E.
195 (OCTAL 303, HEX 12- 3)                 38912.00    40959.00    39909.27     2.56

 SBITS         RATIO..   MIN.     MAX.    H.M.     P.E.          C( 1)..   MIN.       MAX.        H.M.       P.E.

   0                     1.00     1.43    1.18    17.77                    2432.00     9077.47     3836.22    57.74
   1                     1.27     2.85    1.75    38.44                    3142.63    16046.76     5255.93    67.25
   2                     2.55     4.27    3.19    25.25                    6327.06    20485.07     9668.03    52.80
   3                     3.83     5.69    4.58    19.54                    9505.78    24024.48    13621.81    43.30
   4                     5.11     7.11    5.94    16.36                   12683.66    27057.74    17271.21    36.17
   5                     6.39     8.52    7.30    14.32                   17048.85    29754.76    21677.15    27.15
   6                     7.67     9.94    8.66    12.91                   19038.71    32207.44    23931.09    25.70
   7                     8.95    11.36   10.01    11.88                   26153.28    34472.17    29741.97    13.72
   8                    10.23    12.78   11.36    11.09                   29723.53    36586.49    32799.86    10.35
   9                    11.51    14.20   12.71    10.46                   32339.75    38576.91    35184.05     8.80
  10                    12.79    15.61   14.06     9.95                   34507.90    40462.95    37248.92     7.94
  11                    14.07    16.00   14.97     6.42                   36400.86    40959.00    38545.65     5.89
  12                    15.35    16.00   15.67     2.08                   38102.64    40959.00    39479.22     3.61

 ABITS                            AREA..    MIN.        MAX.        H.M.       P.E.
196 (OCTAL 304, HEX 12- 4)                 40960.00    43007.00    41958.55     2.44

 SBITS         RATIO..   MIN.     MAX.    H.M.     P.E.          C( 1)..   MIN.       MAX.        H.M.       P.E.

   0                     1.00     1.29    1.13    12.78                    2560.00     8324.20     3915.76    52.96
   1                     1.15     2.57    1.59    38.26                    2985.91    15745.03     5019.85    68.12
   2                     2.31     3.85    2.89    25.03                    6098.67    20272.24     9376.52    53.75
   3                     3.47     5.13    4.14    19.31                    9384.53    23852.24    13469.55    43.53
   4                     4.63     6.41    5.38    16.12                   12590.56    26908.76    17154.54    36.25
   5                     5.79     7.69    6.61    14.09                   16488.72    29620.60    21184.69    28.48
   6                     6.95     8.97    7.84    12.67                   18975.95    32083.32    23847.25    25.67
   7                     8.12    10.25    9.06    11.64                   22940.34    34355.09    27510.66    19.92
   8                     9.28    11.53   10.28    10.84                   28660.82    36474.45    32098.97    12.00
   9                    10.44    12.81   11.50    10.22                   31784.79    38468.50    34808.71     9.51
  10                    11.60    14.09   12.72     9.71                   34215.75    40357.14    37033.56     8.24
  11                    12.76    15.37   13.95     9.29                   36276.94    42155.51    38995.92     7.50
  12                    13.92    16.00   14.89     6.95                   38098.63    43007.00    40404.29     6.05
  13                    15.08    16.00   15.53     2.95                   39748.85    43007.00    41313.79     3.94
```

Figure B3—Semgent of computer output relating output of IMP hardware to the input data which produced it.

Appendix C

SOME APPLICATIONS OF A PROGRAMMING LANGUAGE

Isobella Cole and E. Steinberger*

SUMMARY

APL is a programming language specifically designed for applied mathematics. It differs from other programming languages in that it is more concise, precise, and economical of symbols. In this paper is shown the APL language as applied to some commonly used transcendental functions. Several improvements of APL are required for such things as program structuring, access to data, presentation of data (i.e., input/output operations), and data operations. However, it is felt that this paper gives some indication of the usefulness of APL if one approaches it openmindedly.

APL is used for such applications as:

1. Trigonometric functions and solution of Kepler's equation

2. Harmonic analysis

3. Matrix operations

The areas of application are chosen primarily for their intrinsic interest to the Mission Trajectory Determination Branch and to indicate the usefulness of the language for Mission Trajectory Determination applications.

TRIGONOMETRIC FUNCTIONS

The APL language for computing the arc cosine, arc sine, arc tangent (principal values), arc tangent (quadrant oriented), sine, and cosine is presented.

*GSFC Programming Systems Branch, M.& T.A.D.

```
      ∇ARCSINCOS[⎕]∇
    ∇ Q←R ARCSINCOS X
[1]   →(R[1]=1)/5
[2]   →(R[1]=0)/17
[3]   'INCORRECT FUNCTION CODE'
[4]   →0
[5]   SIGN←1
[6]   →(X>0)/9
[7]   X←-X
[8]   SIGN←-SIGN
[9]   FX←1.570796305+X×(¯0.2145988016+X×(0.0889789874+X×(¯0.0501743046+X×(0.030891881+X×(¯0.0170881256+X×(
      0.0066700901+X×(¯0.0012624911)))))))
[10]  Q←SIGN×(1.5707963268-((1-X)*0.5)×FX)
[11]  →(R[2]=1)/15
[12]  →(R[2]=0)/0
[13]  'INCORRECT UNITS CODE'
[14]  →0
[15]  Q←57.29577951×Q
[16]  →0
[17]  SIGN←1
[18]  →(X>0)/30
[19]  SIGN←SIGN
[20]  X←-X
[21]  FX←1.570796305+X×(¯0.2145988016+X×(0.0889789874+X×(¯0.0501743046+X×(0.030891881+X×(¯0.0170881256+X×(
      0.0066700901+X×(¯0.0012624911)))))))
[22]  Q1←SIGN×(1.5707963268-((1-X)*0.5)×FX)
[23]  Q←1.5707963268+Q1
[24]  →(R[2]=1)/28
[25]  →(R[2]=0)/0
[26]  'INCORRECT UNITS CODE'
[27]  →0
[28]  Q←Q×57.29577951
[29]  →0
[30]  FX←1.570796305+X×(¯0.2145988016+X×(0.0889789874+X×(¯0.0501743046+X×(0.030891881+X×(¯0.0170881256+X×(
      0.0066700901+X×(¯0.0012624911)))))))
[31]  Q1←SIGN×(1.5707963268-((1-X)*0.5)×FX)
[32]  Q←1.5707963268-Q1
[33]  →(R[2]=1)/37
[34]  →(R[2]=0)/0
[35]  'INCORRECT UNITS CODE'
[36]  →0
[37]  Q←Q×57.29577951
[38]  →0
    ∇
```

Appendix D

UNIFIED INFORMATION PROCESSING TELEMETRY SYSTEM

C. J. Creveling*

Dr. Robert H. Goddard said, "It is difficult to say what is impossible, for the dreams of yesterday are the hope of today and the reality of tomorrow." The system to be described in this paper shows potentialities for approaching the "dream system" discussed by Cliff in a preceding paper of this symposium more closely than might have been suspected possible just a short time ago. Let us first examine a few statistics. There were 18 satellite launches in 1966 and more than 60 are planned between now and 1970, including backups. Second, it should be noted that one of the earliest satellites, the Vanguard, weighed something on the order of 10 pounds while the OGO satellite weighs over 1,000 pounds. There is a spread of two orders of magnitude. Third, bit rates of the earliest Goddard satellites (and even some of the present ones) run as low as 20 bits per second and as high as 128,000 bits per second, or four orders of magnitude higher. The error rate for the sample taken by these satellites at the design goals initially specified runs from 10^{-2} to 10^{-8}, or six orders of magnitude. It is important that some of these facts be kept in mind when adaptive systems are discussed, because no present system can cope with these ranges of variation.

A unified information processing telemetry system will consist, basically, of a programmable computer onboard the spacecraft and an adaptive telemeter in which coding, power, bit rate, and format are controlled. It will include a programmable ground system which will have at least a capability of "talking back" to the satellite. The writer believes the term "adaptive telemetry" has its widest meaning in this sense. A multidisciplinary approach must be considered in order to achieve any solutions to the problems which cover such a gamut. One cannot take the position of a telemetry engineer who considers his system as a common carrier and deals with its inputs and outputs according to some specification.

Figure D1 depicts a system for a simple laboratory experiment that requires only one system engineer — the experimenter. Figure D2 shows

* Goddard Space Flight Center. This is paper No. 9 presented in "Major Space Projects at Goddard," an internal GSFC document (Official Use Only), Jan. 1967.

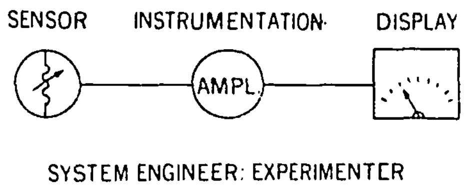


Figure D1— An experiment in the laboratory.

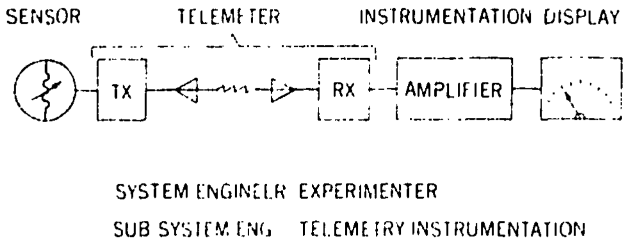

SENSOR
TELEMETER
INSTRUMENTATION DISPLAY
TX
RX
AMPLIFIER
SYSTEM ENGINEER EXPERIMENTER
SUB SYSTEM ENG TELEMETRY INSTRUMENTATION

Figure D2— An experiment on the range.

what happens when the output of a sensor is telemetered from some distance away because of safety considerations or the impossibility of having the experimenter present. Here a second man comes into the picture— the subsystem engineer.

Figure D3 shows an increasing order of system complexity that requires that an increasing number of people become involved in the design. Now the problem can be viewed as a whole and information systems can be discussed. Although data handling inside the subsystems is of concern, the meaning of the data as it passes through the system as a whole is of equal concern. This best illustrates what the term "information" means when an information system is discussed. The overall system has a number of subsystems and it would be very difficult to present in detail adequate specifications of the inputs, outputs, and interfaces between subsystems. Until each subsystem engineer concerns himself with the subsystems which adjoin his, and indeed with the whole satellite plan, adequate control will be lacking. For example, consider an experimenter looking at the pulsed output of a sensor in a simple experiment. Initially, the experimenter would like to see not only the pulse but the wiggles on the pulse to establish in his mind the integrity of the instrumentation. Only after he has been convinced that his experiment is indeed working properly is he interested in the pulse value. If the system gives him this value and nothing else, he will still question the integrity of the experiment. If the experimenter is asked what he would like to have in a telemetry system, he might say, "Give me 5 megacycles!" It is believed that it is possible, with the use of onboard processing and an adaptive system, to satisfy both of these requirements in the sense that they can be controlled from the ground. To do this an initial learning period is provided in which any or all of the experiments can be sampled successively at very high rates in order to establish confidence in the system, and then only the desired information is transmitted.

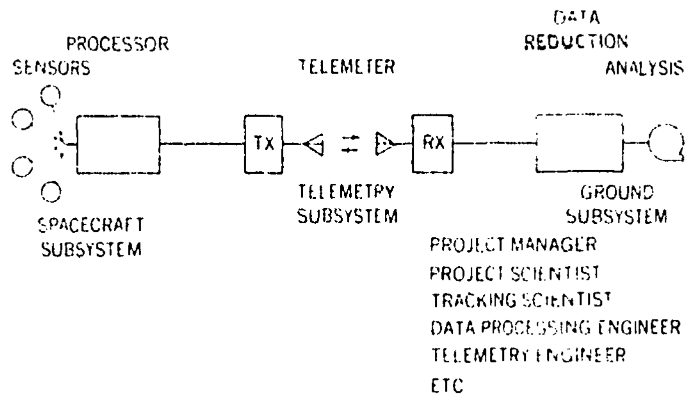


Figure D3— Subsystems used in coupled experiment.

A great deal has been said about the problems of using coding as a technique in various communications systems. Many of these codes, however, have been generated to satisfy specific problems, but our concern here is with the error rates covering six orders of magnitude. It is very difficult to imagine any single set of codes that would satisfy the requirements over this range; however, once there is a computer on board the spacecraft and we have the means for controlling it, the computer becomes a very useful adjunct to coding the telemetry link. Therefore, the computer belongs as much to the telemetry engineer as it does to the onboard processing system.

A system is being proposed by the GSFC Information Processing Division over which they have to exercise a measure of unified control in its design. Unified control means that all of the subsystems will be under a single administration, which will provide a fairly close control over both the administrative and technical problems involved in developing all the various subsystems, from the spacecraft to the ground support. However, when it is necessary to become involved in an interdisciplinary approach, a semantic problem immediately arises between people who at times use the same terms with different meanings, or who use unfamiliar terms. It is difficult, for example, for a person who is not used to working with computers to understand what goes on inside the computer. A thorough understanding of a computer program can be obtained neither by looking at the punch cards nor by looking at the sequence of the machine instruction codes. In fact, it is doubtful that a complete understanding can be obtained by reading the program in some language such as Fortran. To get this understanding, not only must the program be expressed in some relatively higher order language, but also an English description of it and a flow diagram are necessary. A number of so-called higher order languages have been developed for computer programmers in place of, or to supplement, flow charts and the English descriptions which can describe very complicated systems very precisely. For example, the logic structure of the IBM system 360 has been completely specified in a higher order language in approximately a dozen pages. The programming that takes place on the IMP-F satellite, which is a relatively small system, was described by Dr. E. P. Stabler* in this higher order language in a few sentences.

It is presently proposed to study and use such a higher order language for several reasons. First, it is most convenient for conveying information from one subsystem designer to another in a complete and unmistakeable manner. Second, this language lends itself to the writing of the actual programs which will be used in the computer on board the satellite and in the computers on the ground (although they are different types). Third, this language becomes a design tool of the onboard computer because it is possible to write the Boolean expressions for the functions expressed in higher order language (in fact, this has already been started). From these expressions the logic diagrams can be made through a relatively straightforward process.

It is not clear just how useful this language will be to people outside the information system. It would be naive to think that, because this wonderful thing has been developed and discovered, everybody will say, "Teach it to me." What system engineers have to do first is use it for their own benefit. However, these languages have been in vogue and widely distributed now for several years and are finding some use by programmers in general. Moreover, almost all satellite experimenters have been forced to become computer oriented. This provokes the increasing use of these languages as a natural trend.

The kind of machine that is being proposed here for spacecraft computers follows work that was done at GSFC last summer by Dr. E. P. Stabler. At that time, it was undertaken to design a stored program computer that would perform the same functions as one of the IMP spacecraft processors. It was concluded that such a computer be designed with approximately the same size, weight, and power consumption, but it would be busy less than 10 percent of the time. Therefore, it should be possible to build a computer for small satellites that would have a considerable amount of time left for such things as coding the telemetry link, changing the format of the data passing through the

* Associate Professor of Electrical Engineering, Syracuse University.
Such a higher order language now exists and is termed APL.

system, and possibly processing some of the information before it goes into the telemetry link. If there is a computer on board the satellite, a computer on the ground, and a command system, a "dream system" starts to take shape— a dream system in which a spacecraft ultimately might be considered a piece of peripheral gear used in conjunction with a large ground-based computing system. This spacecraft computer would then perform experiment conditioning, multiplex processing, and coding for the telemetry link. In addition, the modularity in design necessary to overcome problems in reliability would enable the configuration to be changed from one mission to another.

Figure D4 shows in a very simple manner a microprogrammed machine and a stored program computer. These differ as shown in Table D1. The conventional computer is characterized by fixed input format and fixed word length. On the other hand, in the microprogrammed machine, variable word length can be provided very inexpensively. The conventional computer has a fixed logic structure, a stored program, and a fixed instruction repertoire that can be set up on the ground. The microprogrammed computer differs in this respect in that the microprograms, as well as the macroprograms, are stored and changeable. In the conventional computer, flexibility is provided by software, and capability for simultaneous operations is limited. In general, . large stored program memory or else a very [illegible]sive list of

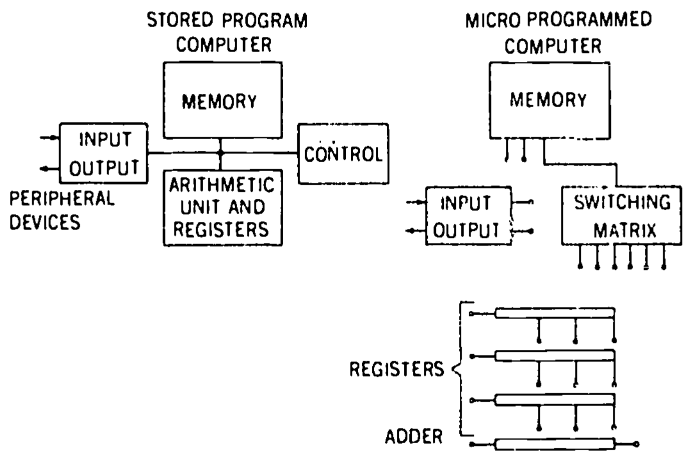


Figure D4— Computer configuration

Table D1— Computer Characteristics

Conventional Computer	Microprogrammed Computer
Fixed input format; fixed word length	Variable input format
Fixed logic structure; stored program and fixed instruction repertoir masking for bit manipulation	Flexible internal structure
Large stored program memory	Small stored program
Low speed (many instructions per macro)	High speed
Flexibility provided by software	
Limited simultaneity	

instructions is required; it is also low-speed in the sense that many machine instructions have to be carried out for each microprogram or so-called macroinstruction. The microprogrammed computer, however, circumvents many of these limitations by being, in effect, almost any machine that you want it to be, and its instructions, in effect, rewire the machine for the problem at hand. The number of instructions to be stored in order to do this is surprisingly modest and the speed is very good.

Development of this system is now progressing in a series of steps. The study phase is in process and a design phase has been started in which the tentative logic designs are being carried out. A breadboard is planned for testing the actual onboard subsystem. It is not necessary to build a computer on the ground because there are a variety of these available and they can be programmed according to higher order language. In order to see how this system will operate as a whole, a computer simulation of the whole system has been planned. This is not going to be an easy job, but it can be accomplished in parallel with the development and later on it will be invaluable in reconfiguring the system according to what is learned from the simulation. Having a flexible onboard computer permits the feedback of both simulation results and early experiences with the system. This will be followed by the development of a prototype and a flight model in the future.

SUPPLEMENT QUESTION AND ANSWER SESSION

UNIDENTIFIED QUESTIONER: I have two things, one is in the nature of a comment. Although the computer is a very useful tool, I think you have to be careful not to use it for things that it is not emminently suited for. For instance, you mentioned perhaps doing a channel coding in the computer, which in certain instances could be a good idea and certainly provides lots of flexibility.

However, I think this is one thing that is quite efficiently done by fairly elegant hard-wired devices which have been developed such as the PN system. To go another step further, if you are counting data pulses from an experiment-interrupt computer and have to add "one" to a register, you could include special accumulators to do this and then transfer the results entirely. You have to be somewhat careful what jobs you do assign to a computer.

The other thing I have is a question about the microprogrammed machine. Do you have any feeling for the difference in complexity between the standard configuration and microprogrammed configuration, especially considering this is a box-labeled switching matrix and may grow to fairly considerable proportion?

MR. CREVELING: I think your remarks are certainly well taken and reflect my feeling on the subject of the software approach versus the hardware approach to specific problems within the spacecraft, such as coding the telemetry link. I mentioned the fact that the computer is available and that the computer is a very flexible device so that as you change from mission to mission, or if you change from very close ranges to very distant ranges as in some of our eccentric satellites, you have the ability then to change your type of coding. It is certainly true that if you have a fixed coding scheme you could build up hardware devices to do this very efficiently and very quickly. In such cases, we have generally followed the practice of investigating both approaches, trying it both ways (at least in the study phase) to decide which is the better, and making the choice on that basis, taking into account the fact that sometimes expediency calls your hand. I feel that the reason that the computer has advantages for doing this is not because of greater efficiency but because of the flexibility of the approach.

Regarding your question as to the difference between the microprogrammed machine and the more conventional machine, we plan to try both of those, and compare them. At the moment, I could not give you a comparison because we have not gotten that far in our study, but I think that the microprogram machine will compare favorably with the conventional approach.

MR. HABIB: In your discussion here a thought occurred to me. I wonder whether we did not miss the point (semantics again) dealing with the word "computer" when we are really talking about general-purpose data or signal-handling system. You began your talk by talking about unified information systems and the designer needed in a sense a control over all of the elements in it. Then we very suddenly get down to a specific thing and call it a computer and I think what flashes into everybody's mind is the type of computer that sits in our floors out here. I think you both were correct that you will use unique equipment words where needed and use general-purpose words where needed and so long as you are allowed to design the entire system, then you will achieve your goals.

MR. CREVELING: No argument.

DR. POSNER: If you were to use your computer to encode the telemetry, I envision that a computer failure in the central computer clock would cut off all information as to what happened, and, in fact, by having all experiments channeled through one machine you may find out that it becomes a no-go instead of an OGO spacecraft. In this case you may be very unhappy at having everything channeled through one point. We at JPL feel that decentralization will protect a mission. Now our missions may be more expensive and you may have to count on getting some use out of them. On a cheaper mission it may be that the reliability is not as important. These are sort of political decisions but in the very expensive heavily equipped spacecraft for soft landing on Mars you have to count on getting some experiments back no matter what fails. So, in a really expensive mission where you have everything staked on it, I would hesitate to think of channeling everything through one computer. On a smaller interplanetary spacecraft where you have dozens of them in the warehouse, you may find it more economical to use the central computer. Incidentally, a possible use for a computer once you have it is to decode the commands from the ground. You may want to have very highly encoded commands to avoid having a miscommand decoded by the spacecraft which might "wipe out" the mission. If you do have a general purpose computer, that would be a really good use for it. Perhaps using a computer to decode spacecraft commands may be a better thing to do than use it to encode telemetry.

MR. CREVELING: That is one of those bad little nightmares that sometimes interrupts our dream system, and we have thought of it. If you will notice in Mr. Purcell's diagram, he shows a switch by which his airborne computing system could, in effect, bypass the data from the commutator (which presumably would remain unimpaired) and would continue to send some kind of data down to the ground. There is another approach, several varieties of approach, to this reliability problem. They generally involve the use of some kind of equipment redundancy. One form which has been used in a large-scale computer is the one in which you have a number of memory cells and a number of computing units, and these two are connected by a matrix. If one or more of these fails, you limp along with something less than the capability of the original system. Since it is programmed, you now reprogram to handle less information. I hope that we can use some such technique to remove the

the fears that you talk about, and, as to your suggestion on using the computer to decode, it is a very good idea.

Appendix E

AN APL LEAST SQUARES PROGRAM

W. P. Altman*

Prior to my arrival at Goddard as a cooperative student from Drexel Institute of Technology in Philadelphia, I had very little contact with computer systems. The previous experience I had, was a course in numerical methods using computers as an aid and having a part-time job in the school's Chemical Engineering Department to reduce sets of given data points to a line of least squares. Even with this experience, I was only using an IBM 1620, a computer with limited memory space and allowable grades of Fortran available. Upon becoming settled in my branch, the Applications Experiment Branch of the Systems Division, I saw a fellow worker sitting before what appeared to be a typewriter with computer printout paper issuing from the back. After I asked what specifically it was, he explained it was a input-output device for the IBM 360/50 computer in Yorktown Heights, New York.

The language used on this computer system was markedly different from Fortran II, the language I was familiar with. Of course, it was APL. After finding out I was too late in signing up for an in-house course, I decided to try to learn the language by myself. I picked up the manual that was supplied with the terminal and sitting down at the terminal, proceeded to become familiar with the system. For a half hour every day I would sign on and experiment with what I had read. Then I would read some more and sign on again and so forth. However, due to the loss of the system because of lack of funds, I have not used the system extensively. Therefore my experiences have been very limited since I have used the APL terminal for only three months. My experience may therefore be of some insight to the beginning user of APL.

One day while using the APL terminal, a fellow employee came into the room and asked what I was doing. He had no previous experience with computers and had no inkling of what a computer could do. He was working on the WEFAX (Weather Facsimile) System, running a statistical test of points trying to evaluate the line of least squares that would fit these points. He asked me to program the APL system to evaluate the coefficients for the line of least squares.

* Drexel Institute of Technology, and GSFC.

Consider a number of points on an XY-coordinate plane. The problem is to develop an equation of a curve that fits these points the "best." "Best" in terms of the least squares criterion means that the sume of the squares of the differences of the data points (y_t) and points on the calculated curve (y_c) are a minimum. The degree of the equation cannot be greater than the number of data points. An nth degree equation (considering one dependent variable) is of the form

$$f(x) = y_c = a_0 + a_1 X + a_2 X^2 \ldots + a_n X^n . \qquad \text{(E1)}$$

One must evaluate the $a_{0,1,\ldots n}$ coefficients. This is done by taking the partial derivatives of function $d(a_{0,1,\ldots n})$

$$d(a_{0,1,\ldots n}) = \sum_{i=1}^{m} (y_t - a_0 - a_1 X - a_2 X^2 - a_n X^n)^2 , \qquad \text{(E2)}$$

with respect to the $a_{0,1,\ldots n}$ coefficients, setting these derivatives equal to zero, and solving the resultant equations simultaneously. The result is

$$\begin{array}{lllll} a_0 N & + a_1 \Sigma_x & + a_2 \Sigma_x^2 & \ldots + a_n \Sigma_x^n & = \Sigma y_t \\ a_0 \Sigma_x & + a_1 \Sigma_x^2 & + a_2 \Sigma_x^3 & \ldots + a_n \Sigma_x^{n+1} & = \Sigma_x y_t \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_0 \Sigma_x^n & + a_1 \Sigma_x^{n+1} & + a_2 \Sigma_x^{n+2} & \ldots + a_n \Sigma_x^{2n} & = \Sigma_x^{n-1} y \end{array} \qquad \text{(E3)}$$

All summations are from 1 to m. Notice that the above equations can be represented by the matrix product (Reference 1):

$$XA = Y, \qquad \text{(E4)}$$

where

$$x_{ij} = \sum_{k=1}^{m} X_k^{i+j-2},$$

$$y_{1,j} = \sum_{k=1}^{m} X_k^{j-1} Y_k, \qquad 1 \le i \le n+1, \qquad \text{(E5)}$$

$$1 \le j \le n+1,$$

$$a_{1,\ell} = a_{\ell-1}, \qquad 1 \le \ell \le n+1.$$

The coefficient matrix A can be found by taking the inverse of X and premultiplying Y. Symbolically

$$A = X^{-1} XA = X^{-1} Y. \qquad \text{(E6)}$$

Trying to make the APL program simple and yet general, I came up with the listing in Figure E1. I designed the program such that I could instruct the fellow I was working for to sign on by himself, load my workspace, enter the data and the degree of the equation required, and call the function "least." This is one of the most important advantages of the APL system— that anyone can sign on and use the system, even someone who has never had a programming course.

As luck would have it, the program was not used extensively. (The need for the program had vanished.) However, the program, when it was used, took a great deal of computer (CPU) time, or the computer was loaded down with other assignments. The reason I say this was that for the evaluation of a fifth degree equation from ten data points, the computer required 36 seconds terminal time. How much of this was CPU time is unknown. The program could have been improved by adding a series of statements to evaluate the function $d(a_{0,1,\ldots n})$ (equation E2) for the prescribed degree function and compare this value for other degree equations.

REFERENCE

1. Southwarth, Raymond W., and Deleeuw, Samual L., Digital Computation and Numerical Methods, McGraw-Hill Book Company, 1965, pp. 472-473.

```
      DATA←0.0 1.0 2.0 0.0 1.0 2.0
       1 LEAST DATA
COEFFICIENTS IN ASCENDING ORDER ARE

 ¯8.881784197E¯16
  1.0000000000E0

     ∇ COEF←DEG LEAST  DATA
[1]    →2+2×((N←0.5×ρDATA)>DEG)
[2]    'DEGREE IS LARGER THAN  NUMBER OF  POINTS TO  BE FITTED'
[3]    →0
[4]    A←(NA,(NA←DEG+1))ρ,0
[5]    XY←(NA,1)ρ,0
[6]    J←0
[7]    →8+7×((J←J+1)>NA)
[8]    I←0
[9]    →10+3×((I←I+1)>NA)
[10]   KK←I+J-2
[11]   A[I;J]←A[I;J]++/((X←DATA[⍳N])*KK)
[12]   →9
[13]   XY[J;1]←XY[J;1]++/((X[⍳N]*(J-1))×(Y←DATA[(⍳N)+N])[⍳N])
[14]   →7
[15]   COEF←(INV A)+.×XY
[16]   'COEFFICIENTS  IN  ASCENDING ORDER  ARE'
     ∇

       1 LEAST 0 1 2  0 1 2
COEFFICIENTS IN ASCENDING ORDER ARE

 ¯8.881784197E¯16
  1.0000000000E0
```

```
     ∇ COEF←DEG LEAST  DATA
[1]    →2+2×((N←0.5×ρDATA)>DEG)
[2]    'DEGREE IS LARGER THAN  NUMBER OF  POINTS TO  BE FITTED'
[3]    →0
[4]    A←(NA,(NA←DEG+1))ρ,0
[5]    XY←(NA,1)ρ,0
[6]    J←0
[7]    →8+7×((J←J+1)>NA)
[8]    I←0
[9]    →10+3×((I←I+1)>NA)
[10]   KK←I+J-2
[11]   A[I;J]←A[I;J]++/((X←DATA[⍳N])*KK)
[12]   →9
[13]   XY[J;1]←XY[J;1]++/((X[⍳N]*(J-1))×(Y←DATA[(⍳N)+N])[⍳N])
[14]   →7
[15]   COEF←(INV A)+.×XY
[16]   'COEFFICIENTS  IN  ASCENDING ORDER  ARE'
     ∇
```

```
       5 LEAST 0,1,2,3,4,5,6,0,3,7,9,13,18,45
        5 LEAST 0,1,2,3,4,5,6,0,2,7,9,13,18,45
COEFFICIENTS IN ASCENDING ORDER ARE

   ¯0.04870129876
    1.79318182
   ¯0.1647727294
    1.092803031
   ¯0.4261363638
    0.04583333334
```

Figure E1

Appendix F

HOW TO FIND PATH LOSS AND SIGNAL/NOISE RATIO USING THE APL TERMINAL

Martin Sachs*

HOW TO FIND PATH LOSS USING THE APL TERMINAL

To compute path loss, or to check for verification of an already known path loss, one may do the following on the terminal:

```
)COPY 1125 DONNA PATHLOSS
)COPY 1125 DONNA LOG
)COPY 1125 DONNA V
```

To execute this function, type the single word PATHLOSS. The terminal will reply with the following:

```
V?KMΔMHZ
⎕:
```

The two variables can then be inserted. The first variable is distance in kilometers (km) and the second is the frequency in megahertz (MHz). A space should be left between the two desired variables. The computer will then type out the answer specified in decibels (db).

For example:

```
)COPY 1125 DONNA PATHLOSS
)COPY 1125 DONNA LOG
)COPY 1125 DONNA V
    PATHLOSS
V?KMΔMHZ
⎕:
      2E5 136
181.1913781
```

* Goddard Space Flight Center.

HOW TO FIND SIGNAL/NOISE RATIO

To compute the signal-to-noise ratio,

```
)COPY 1125 DONNA SNRATIO
)COPY 1125 DONNA LOG
)COPY 1125 DONNA X
```

from the computer. To execute this function, type out the word SNRATIO. The machine will reply with the following:

```
X?DBMΔDKΔBPS
  ENTER VALUES SEPARATED BY COMMAS, AS SHOWN:
  WR (DBM), TN (DEG.K), BR (BITS/SEC.)
⎕:
```

The three variables can then be inserted. The first is WR in dbm (decibels with respect to one milliwatt), the second is temperature (TN) in degrees Kelvin, and the third is bit rate (BR) in bits per second. The answer will then be typed out in decibels (db).

For example:

```
)COPY 1125 DONNA SNRATIO
)COPY 1125 DONNA LOG
)COPY 1125 DONNA X
    SNRATIO
X?DBMΔDKΔBPS
  ENTER VALUES SEPARATED BY COMMAS, AS SHOWN:
  WR (DBM), TN (DEG.K), BR (BITS/SEC.)
⎕:
      ¯135.9,985,100
12.76517921
```

Figure F1 is a sample printout.

```
      )COPY DONNA PATHLOSS

      )COPY DONNA SNRATIO

      ∇ PATHLOSS [⎕] ∇
    ∇ PLS←PATHLOSS;F;R
[1]   'V?KMΔMHZ'
[2]   V←⎕
[3]   F←V[1]
[4]   R←V[2]
[5]   32.5+(20×(10 LOG F))+20×(10 LOG R)
    ∇

      ∇ SNRATIO [⎕] ∇
    ∇ SNRATIO;K;WR;TN;DBM;DK;BPS;BR;SNR
[1]   ' X?DBMΔDKΔBPS'
[2]   ' ENTER VALUES SEPARATED BY COMMAS, AS SHOWN:'
[3]   ' WR (DBM) , TN (DEG. K) , BR (BITS/SEC.)'
[4]   X←⎕
[5]   WR←X[1]
[6]   TN←X[2]
[7]   BR←X[3]
[8]   K←1.38053E¯23
[9]   10×(10 LOG(10*((WR÷10)-3))÷K×TN×BR)
    ∇
```

Figure F1

Appendix G

UNIVAC 1108 UTILITY CONVERSION ROUTINES

Christopher Daly*

During the summer of 1966, my main duties were as a Univac 1108 computer programmer. I was also introduced to APL and had easy access to a nearby terminal. I solved several small analytical problems using APL and wrote a few game-playing routines for my own amusement and to gain proficiency in the language. But perhaps the most generally useful routines were those I wrote in connection with programming the Univac 1108.

A Univac 1108 computer word consists of 36 bits. It is customary, when obtaining memory dumps, to format each word as 12 octal digits. The programmer then has the job of interpreting these 12 octal digits as an instruction, a number, or a s ring of characters. I wrote four short APL functions to aid the programmer in this interpretation:

DECMAL converts a 12-digit octal number into its decimal equivalent. The Univac 1108 uses one's complement notation for negative quantities; this program interprets such quantities correctly.

FLOAT interprets a 12-digit octal number according to the Univac 1108 floating point word format. This format consists of one sign bit, an 8-bit biased characteristic, and a 27-bit mantissa.

DEFLOAT interprets a pair of 12 digit octal numbers according to the Univac 1108 double-precision floating point word format. This format consists of one sign bit, an 11-bit biased characteristic, and a 60-bit mantissa. Although the Univac double-precision format, with its 11-bit biased characteristic, allows representation of numbers between E-308 and E+308, this function will result in a DOMAIN ERROR if interpretation of a number outside the range of E-75 to E+75 is attempted (the smallest and largest numbers that APL can represent.)

FIELDATA converts a string of 12-digit octal numbers into their Univac fieldata equivalent. The result is as though the words were printed directly on the Univac printer.

* Goddard Space Flight Center

I wrote two other APL functions, primarily to assure myself that DECMAL and FLOAT were working correctly, but they are useful in their own right.

OCTAL converts a decimal integer into its 12-digit octal equivalent, as it would appear in a memory dump. It is the inverse function to DECMAL.

FLOCTAL converts a decimal number into its 12-digit octal equivalent in Univac floating point format. For normalized numbers it is the inverse of FLOAT. The composition, FLOCTAL FLOAT, may be used to normalize a number.

The brevity in the statements of these functions is due primarily to the power of the APL base value and representation operators ⊥ and ⊤. The use of these functions is quite simple; even for one floating point conversion it is generally faster to go through the dialing and sign-on procedure to execute FLOAT than it is to convert the number by hand using octal-decimal tables.

Figure G1 is a sample printout.

```
      ∇DECHAL[⎕]∇
    ∇ Z←DECHAL X;Y
[1]    Z←(1 ¯1)[1+X≠Y]×81(12ρ10)⊤Y←X|777777777777-X
    ∇

      ∇FLOAT[⎕]∇
    ∇ Z←FLOAT X;Y
[1]    Z←(1 ¯1)[1+Y≠X]×(81(12ω3)/(12ρ10)⊤Y)×2*¯155+81(12α3)/(12ρ10)⊤Y←X|777777777777-X
    ∇

      ∇DFLOAT[⎕]∇
    ∇ DFLOAT X;[illegible]
[1]    Z←(Y,777777777777-Y)[(¯1+2×C),2×C←1+X[1]≥400000000000]
[2]    M←81(24ω24)/((12ρ10)⊤X[1]),(12ρ10)⊤X[2]
[3]    [illegible] 16
[4]    (1 ¯1)[illegible]×2*¯1084+81(12α4)/(12ρ10)⊤X[1]
[5]    [illegible]
    ∇

      ∇[illegible][⎕]∇
    ∇ [illegible]
[1]    [illegible]
[2]    [illegible]
[3]    [illegible]((ρX)ρ[illegible]+1)/2
    ∇

      ∇OCTAL[⎕]∇
    ∇ OCTAL X
[1]    '01234567654321[illegible]'[1+(7×X<0)+(12ρ8)⊤|X]
    ∇

      ∇FLOCTAL[⎕]∇
    ∇ FLOCTAL X;[illegible]
[1]    [illegible]¯25+1.4426950408889634×⍟|X
[2]    [illegible](8684)/2×[illegible]+|((3ρ8)⊤C+155),(36ρ8)⊤[illegible]
[3]    '01234567654321O'[1+(7×X<0)+[illegible]
    ∇

      DECHAL 777777777[illegible]
¯2[illegible]
      FLOAT [illegible]777777777
0.99999999[illegible]
      DFLOAT 200077777777 777777777777
0.99999999999999996
      FIFLDATA 060710111213 141516170555
ABCDEFGHIJ [illegible]
      OCTAL ¯1
777777777776
      FLOCTAL 3.1415926536
202622077325
```

Figure G1

Appendix H

RADIAN AND DMS

George Fleming*

The trigonometric functions implemented in the APL system use an input in radians. At least occasionally, original data is in the form of degrees, minutes, and seconds of arc, so conversion functions to change from one form to another are desirable. Two such programs are detailed below: (1) *RADIAN*, a function to change degrees, minutes, and seconds of arc into radians, and (2) *DMS*, a function for the reverse. The convenience value of these, especially when used in conjunction with the trig functions, is very great, and extends the usefulness of APL to problems which would be only so much busy work if executed manually.

I. RADIAN: To change degrees, minutes, and seconds of arc into radians

```
∇RAD←RADIAN DEG ∇ RADIAN [⎕] ∇
```

```
[1] DEG←, DEG
```

This insures that the input is availa'`- as a vector.

```
[2] →((ρDEG)=3) ρ THREEIN
```

Input to the function is allowed to be degrees, minutes, and seconds, or minutes and seconds, or just seconds. In the event that ρDEG ≠ 3, it must be changed until ρDEG = 3. For example, an input of 4 seconds may come in as RADIAN 4 or as RADIAN 0 0 4. In the event that ρDEG = 3, control is resumed at statement 6.

```
[3] DEG←0, DEG
```

DEG is increased by one.

```
[4] →((ρDEG)=3)  ρTHREEIN  }
                            } Same as[2]and[3].
[5] DEG←0, DEG             }
```

*GSFC, Processor Development Branch, I.P.D.

```
[6] THREEIN: RAD←0.017453292519943 × DEG [1] + (DEG [2] + DEG [3]
    ÷60)÷60
```

The procedure here is merely to change minutes and seconds into minutes and fractions of a minute, and then change degrees and this new quantity into degrees and fractions of a degree, and multiply by a conversion factor for degrees to radians.

```
        ∇RADIAN[⎕]∇
      ∇ RAD←RADIAN DEG
[1]     DEG←,DEG
[2]     →((ρDEG)=3)ρTHREEIN
[3]     DEG←0,DEG
[4]     →((ρDEG)=3)ρTHREEIN
[5]     DEG←0,DEG
[6]     THREEIN:RAD←0.017453292519943×DEG[1]+(DEG[2]+DEG[3]÷60)÷60
      ∇
```

II. DMS: radians to degrees, minutes, and seconds of arc

```
                              ∇ DMS [⎕] ∇

T60←DMS RAD; DEG; A ; B; C1; C2; C3

[1] C1←0.01745329519943
```

Conversion factor, degrees to radians.

```
[2] C2←0.0002908882086666
```

Conversion factor, minutes to radians.

```
[3] C3←4.848136811E-6
```

Conversion factor, seconds to radians.

```
[4] DEG←(RAD ≥ C1) × (RAD-(A←C1|RAD)) ÷ C1
```

(A) *C1|RAD*: The remainder of *RAD ÷ C1* is computed, so that
(B) it may be subtracted from *RAD*, leaving a quantity (*RAD-C1|RAD*) which is an integer number of degrees, expressed in radians.

(C) This quantity is then converted into degrees by dividing by the conversion factor for degrees to radians.

(D) The above calculations are valid only for values of *RAD* greater than one degree, expressed in radians, hence the (*RAD* ≥ *C*1) factor.

```
[5] DEG -DEG, (A  ≥ C2) ×  (A-(B←C2/A)) ÷ C2
```

The steps here are as in[4],only read "minutes" for "degrees." The final quantity obtained here is in minutes, which is catenated with the value in degrees computed in[4].

```
[6] DEG←DEG, B ÷ C3
```

The last remainder, B, is seconds (expressed in radians), hence the simple conversion and catenation with the calculated degrees and minutes.

[7]→[12] . . . These statements are necessary to compensate for the finite accuracy of the system; in particular, they prevent

```
DMS RADIAN 1 0 0
```

from returning an answer of

```
0 60 0
```

instead of the correct value of 1 0 0. Also prevented is

```
DMS RADIAN 0 1 0
```

returning a value of 0 0 60

```
      ∇DMS[⎕]∇
    ∇ T60←DMS RAD;A;B;C1;C2;C3
[1]   C1←0.01745329251994З
[2]   C2←0.000290888208666
[3]   C3←4.848136811E¯6
[4]   DEG←(RAD≥C1)×(RAD-(A←C1|RAD))÷C1
[5]   DEG←DEG,(A≥C2)×(A-(B←C2|A))÷C2
[6]   DEG←DEG,B÷C3
[7]   →((DEG[3]≠59.9999999725)∧(DEG[3]≠60.0000000118))ρOUT
[8]   DEG[3]←0
[9]   DEG[2]←DEG[2]+1
[10]  →((DEG[2]≠59.9999999725)∧(DEG[2]≠60))ρOUT
[11]  DEG[2]←0
[12]  DEG[1]←DEG[1]+1
[13]  OUT:T60←DEG
    ∇
```

Appendix I

MISCELLANEOUS APL PROGRAMMED FUNCTIONS

William Alford*

APL is useful for design notation, communication (documentation), computation, and programming. Although these categories overlap, the examples chosen here fit primarily into the latter. These programs have been extracted from largerprogram systems because they indicate several distinct features of APL.

Function DIV. This function was written to investigate the repeating decimal series generated from fractions that have prime numbers in the denominator.

```
      ∇ S←N DIV A;R;I;Z;X
[1]     S←'.'
[2]     X←R←A[1]×10
[3]     R←10×R-A[2]×Z←⌊R÷A[2]
[4]     S←S,'0123456789'[Z+1]
[5]     →3×(X≠R)∧N≠¯1+⍴S
      ∇
```

This function demonstrates a method of generating unlimited decimal precision.

```
      10 DIV 1 7
.142857


      10 DIV 1 600
.0016666666


      100 DIV 1 89
.01123595505617977528089887640449438202247191
```

The parameter, N, is the maximum number of decimal places. A is a two-component vector. The first component is the fraction numerator, the

* GSFC Processor Development Branch.

second is the denominator which must be the larger. This function performs the step by step computations of long division. The program terminates when the remainder repeats or when the number of decimal places equals N.

Function COS. The function COS demonstrates the use of vector operations to calculate the Maclaurin Series for the cosine function.

```
    ∇ Z←COS A
[1]   Z←-1--/((3.14159265358979-6.28318530717958|A)*2×⍳13)÷ !2×⍳13
    ∇
```

The parameter A must be in radians. The expression within the inner parenthesis translates any angle into the range between minus and plus pi. Within this range, this expansion is accurate to 14 or 15 decimal places.

Functions BIP1 and MULT 1. These functions (Figure I1) were written for algebraic manipulations. BIP1 will raise a polynomial B to the power A. B is a vector of the polynomial coefficients and A is the scalor power. The coefficients may be in ascending or descending order and the results will be in the same order. This function performs repeated polynomial multiplication using MULT 1.

```
    ∇ C←A BIP1 B;I
[1]   C←,-I←¯1
[2]   →(B≤I←I+1)/0
[3]   →2,C←C MULT1 A
    ∇


    ∇ LIN←V1 MULT1 V2;I;B
[1]   B←(ρ,V1)+¯1+I←ρ,V2
[2]   LIN←V2+.×(I,B)ρV1,Iρ0
    ∇
```

Figure I1

MULT 1 will multiply two polynomials, V1 and V2. This function demonstrates the use of matrix operations rather than the usual loop operation (Figure I2).

```
      1  3  5  6 MULT1  5  1  3  5
5   16   31   49   36   43   30

      1  1 BIP1 5
1   5   10   10   5   1

      1  3  1 BIP1  3
1   9   30   45   30   9   1
```

Figure I2

```
      ∇PROB[⎕]∇
    ∇ PROB
[1]   I←¯0.05
[2]   I←I+0.05
[3]   M←1÷10*I
[4]   P←(100×1-E*-M)÷M
[5]   →(6×I=⌊I)+8×I≠⌊I
[6]   R←S
[7]   →9
[8]   R←Q
[9]   R[⌈P]←'X'
[10]  R
[11]  →2×I<3
    ∇

      E
2.718281828

      S
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------

      Q
|         |         |         |         |         |         |         |         |         |         |

      PROB
```

$$P = 100\,\frac{1-e^{-m}}{m}$$

where $m = \frac{\text{INPUT RATE}}{\text{OUTPUT RATE}}$

Figure I3

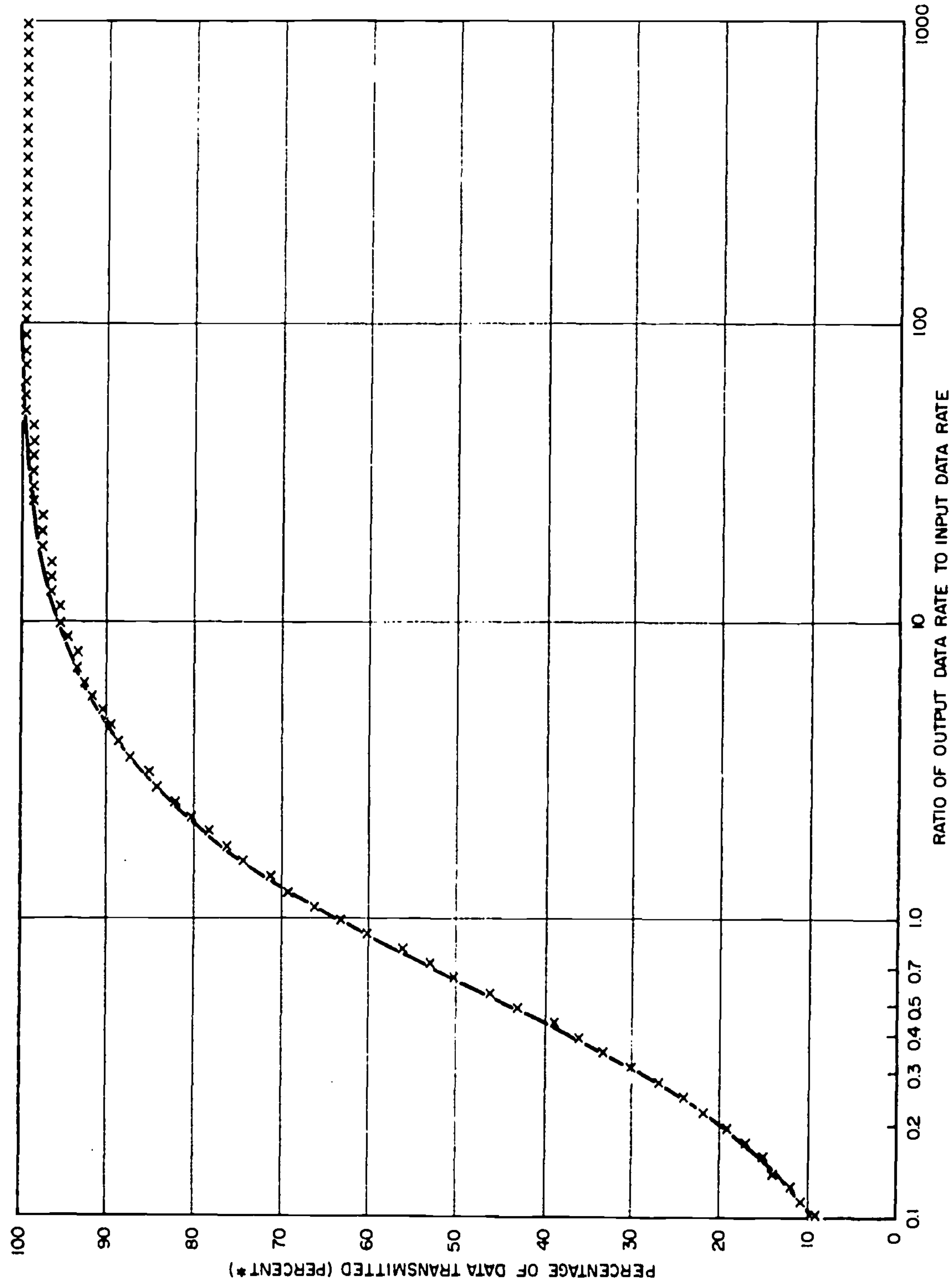

PERCENTAGE OF DATA TRANSMITTED (PERCENT*)
RATIO OF OUTPUT DATA RATE TO INPUT DATA RATE
100
90
80
70
60
50
40
30
20
10
0
0.1
0.2
0.3
0.4
0.5
0.7
1.0
10
100
1000

Figure I4

Appendix J

BAND PASS FILTER DESIGN

Gary Nooger*

Band pass filters are frequently used in many electrical engineering applications. The program discussed here is for a band pass constant-k ladder filter, but the program can easily be extended to include other types of band pass, band elimination, low pass, and high pass filters.

The problem is to design a series of band pass filters with different bandwidths and center frequencies. By use of APL a considerable amount of calculation time was saved.

Figure J1 indicates the input-output terminations of the filter: for a symmetrical filter, the source impedance (Rs) and the load impedance (Rl) are equal.

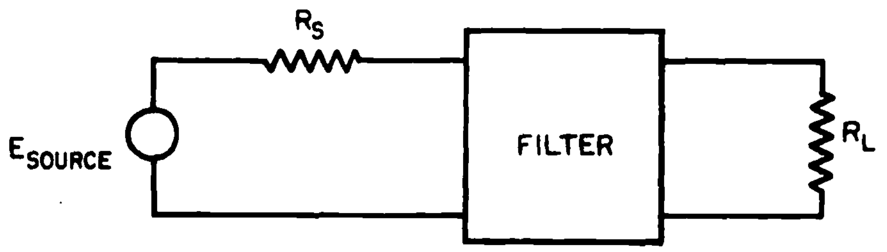


Figure J1

Figure J2 indicates the attenuation and phase characteristics of the filter, where the bandwidth is defined at the -3 db cutoff frequencies and is equal to $f_2 - f_1$.

The program was written so that one can enter the source and load impedance value, the lower 3 db cutoff frequency, and the upper 3 db cutoff frequency. The values for the inductances and capacitances are computed and printed out along with the "T" configuration circuit diagram.

Figure J3 indicated the program listing of the program "BPF."

* Goddard Space Flight Center.

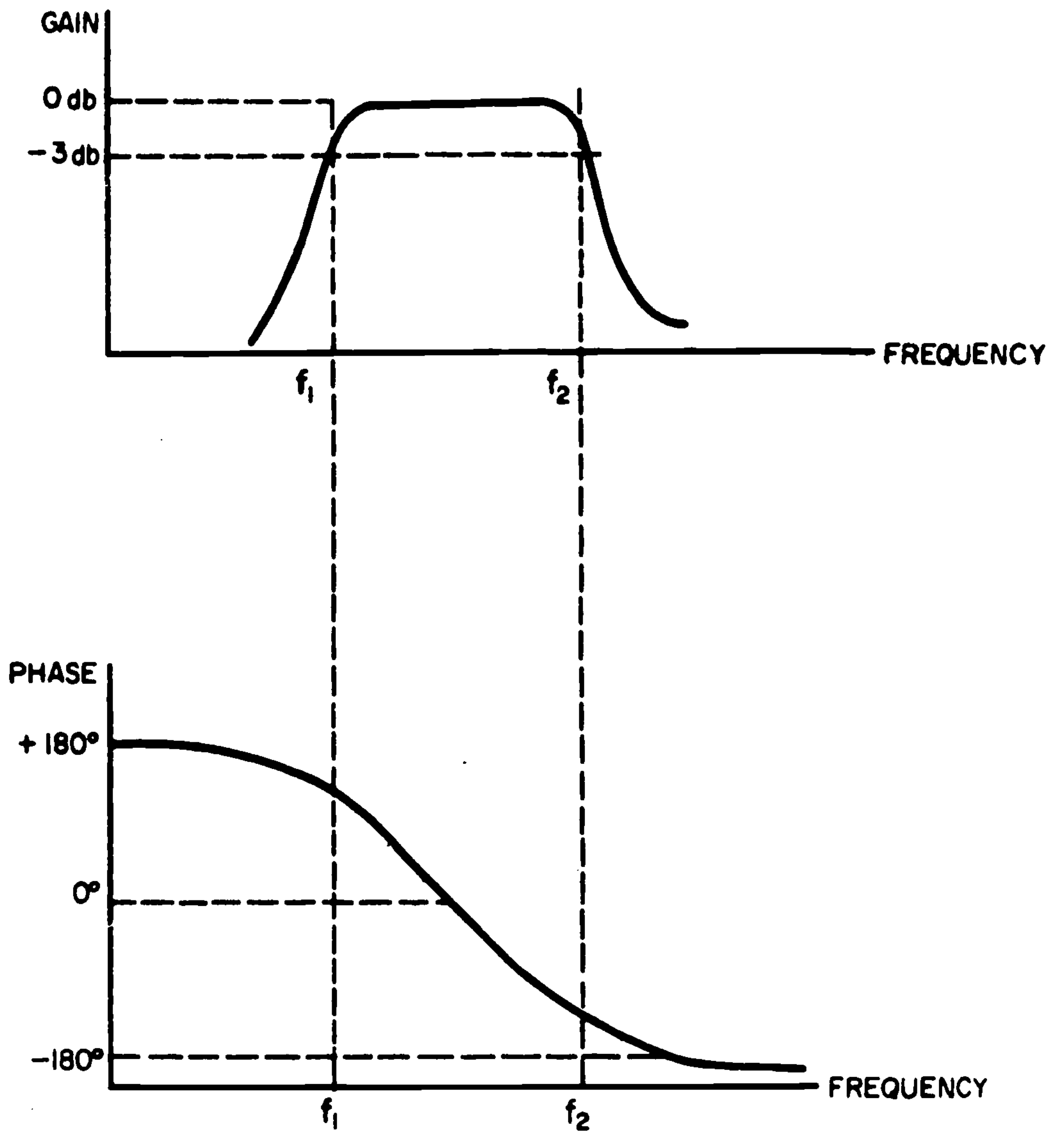


Figure J2

```
    ∇ BPF
[1]   'THIS CIRCUIT IS FOR A BANDPASS CONSTANT-K LADDER FILTER'
[2]   ''
[3]   'TYPE R, UNITS IN OHMS'
[4]   R←⎕
[5]   'TYPE F1 (LOWER 3 DB CUTOFF FREQ), UNITS IN CYCLES'
[6]   F1←⎕
[7]   'TYPE F2 (UPPER 3 DB CUTOFF FREQ), UNITS IN CYCLES'
[8]   F2←⎕
[9]   ''
[10]  →(F1<F2)ρCONTINUE
[11]  'F1 MUST BE LESS THAN F2'
[12]  →0
[13]  CONTINUE:PI←3.141592654
[14]  L1←R÷(PI×(F2-F1))
[15]  L2←(R×(F2-F1))÷(4×PI×F1×F2)
[16]  C1←(F2-F1)÷(4×R×PI×F1×F2)
[17]  C2←1÷(PI×R×(F2-F1))
[18]  ('L1 = ';L1;' HENRY')
[19]  ('L2 = ';L2;' HENRY')
[20]  ('C1 = ';C1;' FARAD')
[21]  ('C2 = ';C2;' FARAD')
[22]  ''
[23]  ('L1÷2 = ';L1÷2;' HENRY')
[24]  ('2×C1 = ';2×C1;' FARAD')
[25]  ''
[26]  '          L1÷2      2×C1            2×C1      L1÷2'
[27]  '  o-------ωωωω-------||--------------||-------ωωωω-------o'
[28]  '                              |'
[29]  '                              |'
[30]  '                        -------------'
[31]  '                        |           |'
[32]  ' ←R→                    ω          ---                   ←R→'
[33]  '                     L2 ω          --- C2'
[34]  '                        ω           |'
[35]  '                        |           |'
[36]  '  o------------------------------------------------------o'
[37]  ''
    ∇
```

Figure J3

Figure J4 shows the result of executing the program with a resistance of 600 ohms, a lower 3 db cutoff frequency of 19,000 Hz, and an upper 3 db cutoff frequency of 21,000 Hz.

```
      BPF
THIS CIRCUIT IS FOR A BANDPASS CONSTANT-K LADDER FILTER

TYPE R, UNITS IN OHMS
⎕:
      600
TYPE F1 (LOWER 3 DB CUTOFF FREQ), UNITS IN CYCLES
⎕:
      19000
TYPE F2 (UPPER 3 DB CUTOFF FREQ), UNITS IN CYCLES
⎕:
      21000

L1 = 0.09549296584 HENRY
L2 = 0.0002393307415 HENRY
C1 = 6.648076152E¯10 FARAD
C2 = 2.652582385E¯7 FARAD

L1÷2  = 0.04774648292 HENRY
2×C1  = 1.32961523E¯9 FARAD

           L1÷2      2×C1           2×C1     L1÷2
 o-------ωωωω-------||--------------||-------ωωωω-------o
                               |
                               |
                     -------------
                     |           |
←R→                  ω          ---                 ←R→
                  L2 ω          ---  C2
                     ω           |
                     |           |
 o------------------------------------------------------o
```

Figure J4

Figure J5 shows the result of executing the program with an illegal condition, i.e., the lower 3 db cutoff frequency is greater than the upper 3 db cutoff frequency. In this case an error message is printed out.

```
      BPF
THIS CIRCUIT IS FOR A BANDPASS CONSTANT-K LADDER FILTER

TYPE R, UNITS IN OHMS
⎕:
      600
TYPE F1 (LOWER 3 DB CUTOFF FREQ), UNITS IN CYCLES
⎕:
      20000
TYPE F2 (UPPER 3 DB CUTOFF FREQ), UNITS IN CYCLES
⎕:
      19000

F1 MUST BE LESS THAN F2
```

Figure J5

Appendix K

A PROGRAMMING LANGUAGE USED TO MANIPULATE AVERAGE OF TELEMETRY DATA

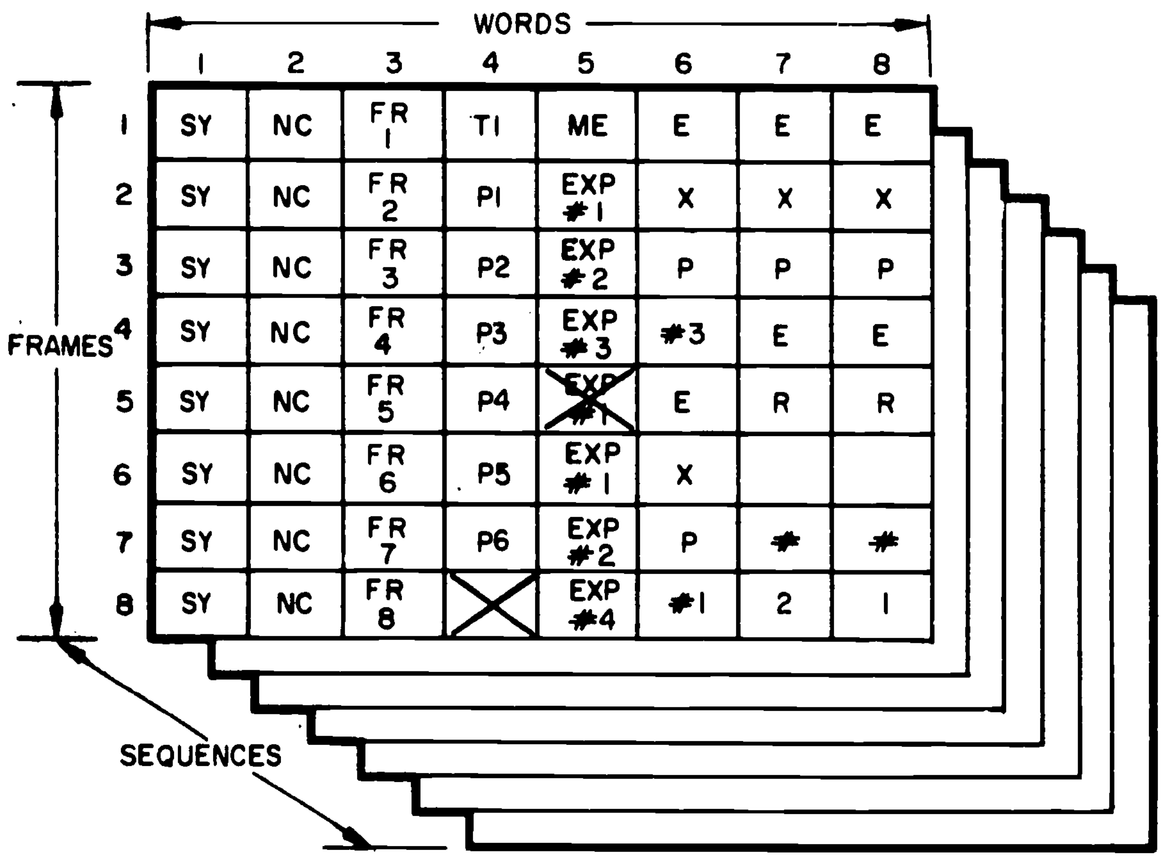
H. Mal Morton* and C. Creveling

The data accumulated by satellites and space probes is telemetered to earth by a radio link, usually as a series of measurements made up by multiplying samples taken successively from a number of sensors. The data also includes certain spacecraft system measurements and timing signals as well. All subsequent manipulation of this data requires a means of recording the data streams and indexing them by their common dimension of time. This dimension also serves to relate the measurements to spacecraft position and orientation in space.

A continuous stream of data is represented by the symbol D, which is a vector of individual measurement in digital form. This vector D is then ordered into a three-dimensional array, in which the first two dimensions represent the words and frames of a telemetry sequence, and the third dimensions becomes a series of sequences.

The utility of this concept, illustrated in Figure *K*1 is that the standard APL indexing and operators can be used to index, sort, assemble, reorder, and manipulate the data. A few of these are shown in the expressions accompanying Figure *K*1.

* I. B. M. and GSFC


WORDS
1 2 3 4 5 6 7 8
FRAMES
1 2 3 4 5 6 7 8
SY NC FR 1 TI ME E E E
SY NC FR 2 PI EXP #1 X X X
SY NC FR 3 P2 EXP #2 P P P
SY NC FR 4 P3 EXP #3 #3 E E
SY NC FR 5 P4 EXP #1 E R R
SY NC FR 6 P5 EXP #1 X
SY NC FR 7 P6 EXP #2 P # #
SY NC FR 8 EXP #4 #1 2 1
SEQUENCES

```
ARRAY OF TELEMETRY DATA
                                D ←(8,8,SEQ)ρ ''TELEMETRY DATA STREAM''
ARRAY OF SYNC PATTERNS
                                SYNC←(8α2)/ [2] D
MATRIX OF TIMES
                                TIME←D [1;4,5;]
MATRIX OF EXPERIMENTER # 1's DATA (INCLUDING TIME)
                                EXPERI←D [;8;],[1] D [1;4,5;],[1] D [2;5;]
```

Figure *K*1— Array of satellite telemetry data.