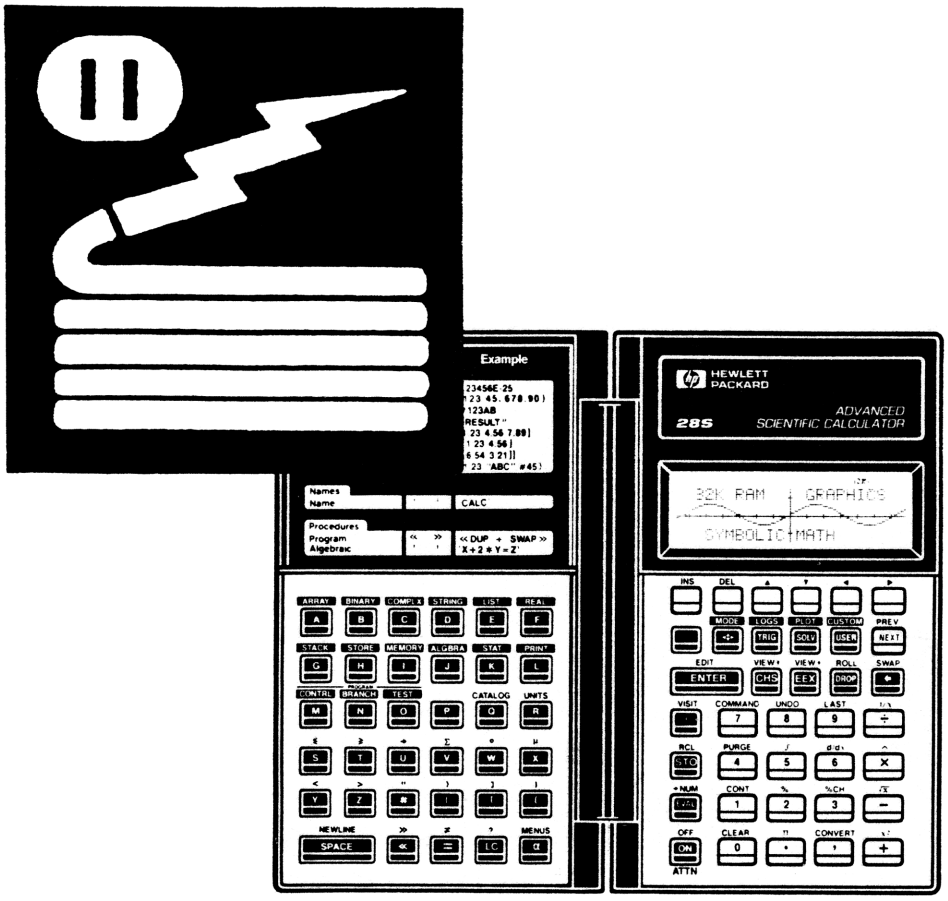
HP-28S
Engineering Applications

by Robert Boyd



EduCALC
27953 Cabot Road
Laguna Niguel, CA 92677 U.S.A.

Stock #2219

HP-28S ENGINEERING APPLICATIONS

by

Robert R. Boyd

EduCALC
27953 Cabot Road
Laguna Niguel, CA 92677

(714) 582-2637

ACKNOWLEDGEMENT:

Many thanks to friend and colleague Gary Garth for reviewing the manuscript and providing invaluable corrections and comments.

DISCLAIMER:

The author and/or EduCALC, Inc. make no warranty of any kind with regard to this material. The author and/or EduCALC, Inc. shall not be liable for errors contained herein or for inicidental or consequential damages in connection with the furnishing, performance, or use of this material.

TECHNICAL SUPPORT:

For the price of a self-addressed stamped envelope (SASE), the author will answer questions and provide help for programs that will not run properly. This will generally take the form of a printout of the author's working program listing, plus inputs, outputs, flag status, plus suggestions and comments. Send questions and SASE to:

HP-28S HELP

P.O. Box 5950-260

Orange, CA 92667

PREFACE

These programs have been found useful over the years, first as HP-41 and then as HP-71B programs. Since Hewlett-Packard has regrettably discontinued the HP-71B, they were translated for the HP-28S. It is hoped they will have some practical value to the reader of this book. They are unavoidably slanted towards electrical engineering, but at least half of the subjects should be useful to all engineering disciplines.

Some methods are original with the author, but most programs were adapted from various technical references which are provided. No attempt has been made to optimize the programs. The interested reader will see a faster or more efficient way to do it.

The programs are "bare bones", i.e., for the most part there are no input or output label strings. These embellishments are left for the reader to add after gaining familiarity with the programs. The HP-28S characters π and $\sqrt{\ }$ are shown as "pi" and "sqrt", respectively.

The programs are presented in the following format:

a. Introduction:

   An explanation of what the program is used for. Some exposure to the material by the reader is assumed.

b. Stack diagram:

   A stack diagram is used to show the stack contents before and after program execution.

c. Examples:

   One or two examples are given so the user can check the program to make sure it was entered correctly.

d. Listing:

   Commented listings of the programs.

e. References:

   This section will either give book or article reference(s). If the method is original, a theoretical explanation is presented.

TABLE OF CONTENTS

A WORD ABOUT DIRECTORIES

The author takes advantage of the directory scheme in the HP-28S by having the root directory (HOME) contain directories of major categories of programs, such as circuit analysis, math, signal processing, etc.

The HOME directory is designated as level 1, the next level down level 2, etc., Level 4 is the deepest level. Level 4 is used to store constants that are used by programs in level 3.

Thus the programs will include directory changes in the beginning to access the constants in the level below such as << LVL4 'A' STO A ... UP >>. The CRDIR command is used to create the downward directory changes.

The command UP is an upward directory change to get back to level 3 or higher directories. Procedure UP would be: << next level up name >>. As an example, a specific directory path is given below:

Path = { HOME MATH MAT LVL4 }

Downward directory changes from HOME created by 'CRDIR'	Upward directory changes created by programs
MATH (Goes to level 2 when executed)	<< HOME >> 'LVL1' STO
MAT (Goes to level 3)	<< MATH >> 'LVL2' STO
LVL4 (Goes to level 4)	<< MAT >> 'UP' STO

Another directory path has only three levels:

Path = { HOME SIGPR LVL3 }

Downward directory changes from HOME created 'CRDIR'	Upward directory changes created by programs
SIGPR (Goes to level 2 when executed)	<< HOME >> 'LVL1' STO
LVL3 (Goes to level 3)	<< SIGPR >> 'UP' STO

The user may of course create his/her own directory system. These names are given here so the reader can follow the program listings.

I. STATE SPACE METHODS

1. State Transition Matrix

a. Introduction:

Leverrier's Algorithm provides a straightforward means of obtaining the state transition matrix (STM) from the following matrix equation:

$$dx/dt = Ax \qquad (1)$$

where A is an N x N matrix of coefficients and x is an N x 1 column vector. The STM is defined as:

$$(sI - A)^{-1} \qquad (2)$$

where I is an N x N identity matrix and s is the Laplace transform variable.

The STM initially appears as a transfer function in s, with the denominator being the characteristic equation, and the numerator a matrix polynomial. When matrix elements are collected in the numerator, the final form of the STM results. The roots of the characteristic equation are then the eigenvalues of the A matrix. (See example below.)

b. Stack diagram: (- indicates empty.)

Level	Before	After
N + 2	-	$s^{(N-1)}$ matrix
N + 1	-	$s^{(N-2)}$ matrix
.	.	.
.	.	.
.	.	.
3:	-	s^0 matrix
2:	-	"Denom coeff are"
1:	A matrix	[Coeff vector T1]

c. Example:

```
                [[ -1   0  0 ]
Let matrix A = [  0  -4  4 ]    (N = 3)
                [  0  -1  0 ]]
```

Key in the matrix and enter into level 1: [[-1 0 0 [0 -4 4 [0 -1 0 ENTER. Press SIMA (S I Minus A)

The stack will contain the following five objects when the program has finished: (About 4 seconds execution time.)

```
5:  [[ 1  0  0 ]
     [ 0  1  0 ]      (matrix coefficient of s²)
     [ 0  0  1 ]]

4:  [[ 4  0  0 ]
     [ 0  1  4 ]      (matrix coefficient of s¹ = s)
     [ 0  -1 5 ]]

3:  [[ 4  0  0 ]
     [ 0  0  4 ]      (matrix coefficient of s° = 1)
     [ 0  -1 4 ]]

2:  "Denom coeff are"

1:  [ 5  8  4 ]
```

From this we obtain:

$$\frac{\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} s^2 + \begin{vmatrix} 4 & 0 & 0 \\ 0 & 1 & 4 \\ 0 & -1 & 5 \end{vmatrix} s + \begin{vmatrix} 4 & 0 & 0 \\ 0 & 0 & 4 \\ 0 & -1 & 4 \end{vmatrix}}{s^3 + 5s^2 + 8s + 4}$$

The STM is then:

$$\frac{\begin{vmatrix} s^2 + 4s + 4 & 0 & 0 \\ 0 & s(s+1) & 4(s+1) \\ 0 & -(s+1) & s^2 + 5s + 4 \end{vmatrix}}{s^3 + 5s^2 + 8s + 4}$$

d. Listing:

Program listing	Comments
'SIMA'	Main program
<< LVL4 'A' STO A SIZE LIST-> DROP2	Get A matrix & order n
-> n << n 1 ->LIST 0 CON 'T1' STO	Scratch for denominator coefficients
n IDN 'D1' STO	Create identity matrix D1
n LALG	Subprogram LALG (Leverrier's ALGorithm)
"Denom coeff are" T1	Label T1 vector
'C1' 'T1' 'T' 'D1' 'F' 5 ->LIST	
PURGE UP >> >>	Purge and go back up to LVL3 subdirectory
'UP'	
<< MAT >>	Upward directory change to level 3
'LALG'	
<< -> n << 1 n FOR q	Start algorithm loop
IF 1 q SAME THEN D1 'F' STO F END	Start with identity matrix
A F * 'C1' STO	Multiply A times F
0 'T' STO n q GETT	Subprogram GETT (GET T)
IF q n ≠ THEN 'T1' q GET D1 *	
C1 + 'F' STO F END NEXT >> >>	Put matrix F on stack and repeat.
'GETT'	
<< -> n q << 1 n FOR i 'C1' i DUP 2	
->LIST GET 'T' STO+ NEXT	Get trace of C1 = A * F
'T1' q T q / NEG PUT >> >>	Store denom coeff.

e. References:

1. State Space and Linear Systems, Donald M. Wiberg, Schaum's Outline Series, 1971, p.102.

The alogrithm is given as:

Step 1: $F_1 = I$, $T_1 = -trAF_1/1$

Step 2: $F_2 = AF_1 + T_1 I$, $T_2 = -trAF_2/2$

.
.
.

Step n: $F_n = AF_{n-1} + T_{n-1} I$, $T_n = -trAF_n/n$

$$\text{Then } (sI - A)^{-1} = \frac{s^{n-1}F_1 + s^{n-2}F_2 + \ldots + sF_{n-1} + F_n}{s^n + T_1 s^{n-1} + \ldots + T_{n-1}s + T_n}$$

where tr denotes the trace (sum of diagonal elements), A, F are matrices, and T is a scalar.

2. Characteristic Equation

a. Introduction:

A faster method of finding the characteristic equation (CE) is Bocher's Formula. If the complete STM is not needed, then this series of programs is preferred. The roots of the CE (eigenvalues) can be found from the programs in III.1.

b. Stack Diagram:

Level	Before	After
1:	A matrix	[Vector of CE coefficients]

c. Examples:

Using the same example A matrix as in I.1. above, key in the matrix and enter into level 1: [[-1 0 0 [0 -4 4 [0 -1 0 ENTER. Press BOCH (BOCHer's formula). When the program has finished, level 1 will show:

1: [1 5 8 4]

Note that the coefficient of the highest power of s, which will always be unity, is included. In Leverrier's Algorithm in I.1., the unity coefficent is assumed.

Hence the CE is: $s^3 + 5s^2 + 8s + 4$.

Second example: Key in [[2 -2 3 [1 1 1 [1 3 -1 ENTER, BOCH; see in level 1: [1 -2 -5 6]. The CE of this matrix is $s^3 - 2s^2 - 5s + 6$.

d. Listing:

Program Listing	Comments
'BOCH'	
<< LVL4 'A2' STO A2 DUP 'A1' STO	Save matrix A2 in A1 for powers of A calculation.
SIZE LIST-> DROP2 -> n << n 1 +	Get order N of A matrix
1 ->LIST 0 CON 'B' STO	Scratch for coefficients
n 1 ->LIST 0 CON 'T' STO	
1 n FOR i A1 n TRACE	Get trace of A matrix
'T' i ROT PUT 'A1' A2 STO*	Get A^2, A^3, etc.
NEXT 'B' 1 1 PUT 'T' 1 GET NEG	
'B' 2 ROT PUT 2 n FOR i i 1 +	Start Bocher's sequence
-> q << 'B' q 0 PUT i 'K' STO	Auxiliary index K
1 i FOR j 'B' K GET 'T' j GET	
* 'B' q GET + 'B' q ROT PUT 'K'	
1 STO- NEXT	Decrement K and repeat
'B' q GET i / NEG 'B' q	
ROT PUT >> NEXT >> B	Put vector B on stack
'K' 'A1' 'T' 'B' 4 ->LIST	
PURGE UP >>	
'TRACE'	
<< 0 -> a1 n c << 1 n FOR i 'a1'	
i i 2 ->LIST GET c + 'c'	Get diagonal sum
STO NEXT c >> >>	Pass sum back to BOCH

e. Reference:

1. State Variables for Engineers, DeRusso, Roy, & Close, John Wiley & Sons, 1965, p. 234.

3. Transfer Matrix

a. Introduction:

State space analysis leads to the transfer matrix, which is very useful for multiple-input/multiple-output automatic control system analysis. It is also used for circuit analysis and is especially useful in the analysis of switched mode power supplies (SMPS). Using an averaging technique called state space averaging, many SMPS small signal transfer functions such as input and output impedance can be obtained.

To obtain the transfer matrix, we begin with the following two sets of matrix equations:

$$dx/dt = Ax + Bu \qquad \text{(system equation)} \quad (3)$$

$$y = Cx + Eu \qquad \text{(output equation)} \quad (4)$$

where y and u represent output and input vectors respectively, and x is an N x 1 vector of unknowns.

For the dimensions, let N be the system order, M the number of inputs, and K the number of outputs. Then we have:

matrix A is N x N	vector x is N x 1
matrix B is N x M	vector u is M x 1
matrix C is K x N	vector y is K x 1
matrix E is K x M	

After taking the Laplace transform, (3) and (4) become:

$$sx(s) = Ax(s) + Bu(s) \quad (5)$$

$$y(s) = Cx(s) + Eu(s) \quad (6)$$

Solving (5) for x(s):

$$x(s) = (sI - A)^{-1}Bu(s) \quad (7)$$

Substituting into (6) and dropping the (s) functional notation:

$$y = C(sI - A)^{-1}Bu + Eu \quad (8)$$

If we let the STM = $Q = (sI - A)^{-1}$ (which can be found from Leverrier's algorithm in I.1.) and divide by u we have the transfer matrix:

$$y/u = G = CQB + E \quad (9)$$

Note that CQB must have dimensions of E or K x M to be conformable for addition. Hence the transfer matrix G will also have dimensions of K x M, or no. of outputs x no. of inputs. Reinstating the (s) notation, G can be given generally as:

$$G(s) = \begin{vmatrix} G_{11}(s) & G_{12}(s) & \cdots & G_{1M}(s) \\ G_{21}(s) & G_{22}(s) & \cdots & G_{2M}(s) \\ \vdots & \vdots & & \vdots \\ G_{K1}(s) & G_{K2}(s) & \cdots & G_{KM}(s) \end{vmatrix}$$

where $G_{IJ}(s) = \dfrac{\text{output I}}{\text{input J}}$.

b. Stack Diagram:

Level	Before	After
N + 2	-	$s^{(N-1)}$ matrix
N + 1	-	$s^{(N-2)}$ matrix
.	.	.
.	.	.
.	.	.
4:	A matrix	s^1 matrix
3:	B matrix	s^0 matrix
2:	C matrix	"Denom coeff are"
1:	E matrix	[Coeff vector T1]

c. Examples:

For the first example, let N = 3, K = M = 1;

```
                [[ 0  1  0 ]
let matrix A =   [ 0  0  1 ]
                 [ -30 -31 -10 ]]
```

let the B matrix, or in this example, the b column vector be
(note lower case to denote vectors, upper case to denote matrices.)

```
     [[ 0 ]
b =   [ 0 ]  ;
      [ 1 ]]
```

let the row vector c = [[4 5 1]];

and let e be the null vector [[0]].

Key in these arrays in A, b, c, e order: [[0 1 0 [0 0 1
[30 CHS 31 CHS 10 CHS ENTER [[0 [0 [1 ENTER [[4 5 1 ENTER
[[0 ENTER. Then press CQB. The stack should show at completion:

5: [[1]]

4: [[5]]

3: [[4]]

2: "Denom coeff are"

1: [10 31 30]

To interpret these answers, remember that K = M = 1, and the
transfer matrix G will have dimensions of K x M or 1 x 1. Hence

$$G(s) = G_{11}(s) = \frac{s^2 + 5s + 4}{s^3 + 10s^2 + 31s + 30}$$

As a more interesting example, key in the following arrays, again
in A, B, c, e order:

```
     [[ -2.2  -0.6 ]
A =   [ 0.6  -0.2 ]]

     [[ 1 -1 0 0 ]
B =   [ 0  0 0 0.6 ]]

c = [[ 1.2  0.6 ]]

e = [[ 0 0 0 1.2 ]]
```

Here N = 2, K = 1, and M = 4. When CQB is executed the stack should be:

4: [[1.2 -1.2 0 1.56]]

3: [[.6 -.6 0 1.56]]

2: "Denom coeff are"

1: [2.4 .8]

Now G(s) is 1 x 4 and if we let $D(s) = s^2 + 2.4s + 0.8$, the transfer matrix elements are:

$G_{1,1}(s) = (1.2s + 0.6)/D(s)$

$G_{1,2}(s) = -(1.2s + 0.6)/D(s)$

$G_{1,3}(s) = 0/D(s) = 0$

$G_{1,4}(s) = 1.56(s + 1)/D(s)$

d. Listing: (This is a short program, as most of the work is done by the SIMA program.)

Program Listing	Comments
'CQB'	
<< -> a b c e << a SIMA a SIZE LIST->	Fill stack with Q matrices
DROP2 -> n << n NTP n NTP	NTP is a utility subroutine
1 n START c SWAP * b * e +	Form G = CQB+E
n NTP NEXT >> >> >>	Roll down stack
'NTP'	
<< -> n << n 2 + ROLLD >> >>	

e. References: See I.1. and I.2.

II. POLYNOMIALS

1. Real or Complex Roots, n > 4

a. Introduction:

As the title conveys, this series of programs solve for the real and/or complex roots of polynomials with real coefficients of order > 5. Reference 1 has some excellent programs concerning polynomials, among which are programs that will solve for real /complex roots for orders ≤ 4. If your requirements never exceed order 4, then it is suggested that the programs from reference 1 be used and this section omitted. The programs in reference 1 are explicit closed form solutions (for up to fourth order) and execute faster.

The author has tested the programs given here up to order 25. Although the programs will solve some equations with multiple roots, the user is cautioned not to use these programs for multiple roots. Since identical roots rarely occur in practice, this restriction should not be too confining.

Equations with the coefficient of the highest power of s other than one are permitted, as the program divides through by this coefficient if it is not unity.

The method used is known as Bairstow's method. (Reference. 2.) (AKA Lin-Bairstow method.) The method pulls out the quadratic factors from a polynomial with real coefficients and solves that quadratic for real or complex roots. Thus no root searching is done in the complex plane.

The roots are displayed in complex form, whether they are real or complex. Roots given such as (-52, 1E-19) should be assumed real, i.e., the imaginary part may not be precisely zero.

The program is started by entering a LIST of polynomial coefficients and pressing BEGN. The level of precision is determined by the display mode. That is, if the display mode is (FIX, SCI, or ENG) n, then the precision is $1/10^n$. If in STD mode, the precision defaults to 0.001 (FIX 3).

b. Stack diagram:

Level	Before	After
n	-	(A_n, B_n) nth root
n - 1	-	(A_{n-1}, B_{n-1})
.	.	.
.	.	.
.	.	.
2:	-	(A_2, B_2)
1:	(list of coefficients)	(A_1, B_1)

c. Examples:

In the (list of coefficients), the order of the polynomial coefficients is in descending powers of the variable, from left to right.

The first example will be an easy fifth order polynomial made up of integer roots:

$$P(s) = (s + 1)(s + 2)(s + 3)(s + 4)(s + 5)$$

$$= s^5 + 15s^4 + 85s^3 + 225s^2 + 274s + 120$$

which is shown in factored form first so we can identify the roots to check the program. Enter P(s) in list form as follows:

(1 15 85 225 274 120)

in FIX 3 mode and press 'BEGN'.

During program execution, display levels 1 and 2 will show the iterations of u and v in the quadratic $s^2 + us + v$. Since P(s) in this example is fifth order, the first real root will be found and then the search for u and v starts. After u and v are found, the deflated polynomial remaining is second order and the 2nd u and v are given ipso facto.

(The reader is reminded of the fact that any odd order polynomial has at least one real root.)

After about 30 seconds of run time the stack should show:

```
5:   (-4.004,0.000)
4:   (-4.998,0.000)
3:   (-2.000,0.000)
2:   (-2.998,0.000)
1:   (-1.000,0.000)
```

By increasing the display mode to FIX 6, the roots will be exact

The second example will be a 7th order polynomial, first given in factored form so the roots can be identified, and then in polynomial form.

Let $P(s) = (s + 30)(s + 500)(s + 9000)[(s + 150)^2 + 300^2]$

$[(s + 75)^2 + 80^2]$ which when multiplied out:

$$P(s) = s^7 + 9980s^6 + 9243025s^5 + 3924305750s^4 +$$

$$1.0684781625E12s^3 + 1.33786940625E14s^2 +$$

$$9.2383453125E15s + 1.826296875E17$$

Thus we are looking for three real roots, and two complex conjugate pairs, which gives us the required seven.

Key in the polynomial in list form: { 1 9980 9243025 3924305750

1.0684781625E12 1.33786940625E14 9.2383453125E15 1.826296875E17 }

and ENTER into level 1 and press BEGN to start the program.

After program completion, the stack will show (FIX 3 format):

```
7:          (-500.051,0.000)
6:          (-9000.000,0.000)
5:          (-149.997,-300.025)
4:          (-149.977,300.025)
3:          (-74.998,-79.999)
2:          (-74.998,79.999)
1:          (-30.000, 0.000)
```

which are close to the expected roots. Again, by using a larger display mode, e.g., FIX 5, the roots will be closer to exact. The user is cautioned about using an excessively large display mode as this can cause run times to be very long. It is best to start with a relatively small display mode and work up.

d. Listing:

Program Listing	Comments
'BEGN'	Main program
<< LVL4 59 CF 'PE' STO GFIX	Store Polynomial Equation & get display mode.
( ) 'RTL' STO 5 CF PE SIZE -> n <<	
PE 1 GET 1 IF ≠ THEN n NORM END	Divide by coeff of s^n
n 2 / FP 0 IF == THEN RROOT 5 SF END	If odd order, get real root.
PE SIZE 3 IF > THEN	Minimum order is n = 3.
DO 0 0	Begin outer loop with initial estimates of u and v.
DO PE GSR GUV DUP2 CLLCD	Get (remainders) S & R; Get U & V.
1 DISP 2 DISP UNTIL 1 FS? END	Display iterations
RTL + + 'RTL' STO QT LIST-> 3	RTL is the RooT List QT is the QuotienT list
DROPN DEPTH ->LIST 'PE' STO	Store deflated polynomial
UNTIL PE SIZE 3 == END END	Continue until one quadratic with 3 coefficients left.
PE LIST-> DROP RTL + + 'RTL' STO	
CLEAR CLMF (DA QT S PE) PURGE RTS	Get quadratic roots
IF 5 FS? THEN RRL LIST-> DROP	Display odd order real root
SWAP DROP NEG 0 R->C 'RRL' PURGE END	RRL is Real Root List
UP >> >>	
'RROOT'	Real ROOT routine
<< 35 36 SF SF PE 'S' PVAL	PVAL & PDIV from reference 1.
'S' -1 ROOT NEG 1 SWAP 2 ->LIST	Get real root using HP-28S firmware routine ROOT.
PE SWAP PDIV 'RRL' STO DROP	Store in Real Root List.
'PE' STO >>	Store deflated even order polynomial.

```
'GSR'                                   Get (remainders) S & R routine.

<< -> u v num << num SIZE -> n          Store u & v estimates and
                                        polynomial list 'num'.
<< num 'QT' STO num 1 GET DUP           Start synthethic division.
                                        (See reference 2.)

QT 1 RPQS u * NEG                       QT is QuotienT list.

num 2 GET + QT 2 RPQS                   Utility routine RPQS

3 n 2 - FOR i num i GET QT i 1 -        Continue synthethic division

GET u * - QT i 2 - GET v * - QT i RPQS

NEXT num n 1 - GET QT n

2 - GET u * - QT n 3 - GET v * - QT n

1 - RPQS QT num n GET SWAP n 2 - GET v

* - QT n RPQS u v >> >> >>


'RPQS'                                  Utility subroutine

<< ROT PUT 'QT' STO >>


'GUV'                                   Get U & V

<< QT 'DA' STO -> u v <<                Space for D Array and pass
                                        u & v.
QT 1 GET DUP DA 1 RPD                   Utility routine RPD

u * NEG QT 2 GET + DA 2 RPD QT

SIZE -> n << 3 n 2 - FOR j              Begin u & v calculations

QT j GET DA j 1 - GET u * - DA

j 2 - GET v * - DA j RPD NEXT QT n 2 -

GET DA n 4 - GET v * - DA n 1 - RPD DA

n 3 - GET v * NEG DA n RPD DA n 2 - GET

DA n 1 - GET * DA n 3 - GET DA n GET    Get den = denominator

* - -> den << DA n 1 - GET QT n 1 -
```

```
GET * DA n 3 - GET QT n GET * - den /
DA n 2 - GET QT n GET * DA n GET QT
n 1 - GET * - den / -> du dv <<          Get du and dv
u v du dv 1 CF CALE                     CALculate Error, du/u - ER
IF 1 FS? THEN u v ELSE u du +           u -> u + du, v -> v + dv
v dv + END >> >> >> >> >>

'RPD'                                   Utility subroutine
<< ROT PUT 'DA' STO >>

'CALE'                                  CALculate Error
<< -> u v du dv << IF du u /
ABS ER < dv v / ABS ER < AND
THEN 1 SF END >> >>                     Set flag 1 if both less than
                                        error input ER.
'RTS'                                   Get RooTS
<< RTL LIST-> 2 / -> m
<< 1 m START QUD RND SWAP RND           Solve quadratics
m 2 * ROLLD m 2 * ROLLD
NEXT >> 'RTL' PURGE >>

'PVAL'                                  Polynomial VALue
                                        Copyright Hewlett-Packard Co.
<< -> st x << st 1 GET 2 st SIZE FOR    Used with permission.
                                        See reference 1.
n x * st n GET + NEXT >> >>
'QUD'                                   QUaDratic solve
                                        Copyright Hewlett-Packard Co.
<< SWAP 2 / NEG DUP SQ ROT - sqrt       Used with permission.
                                        See reference 1.
DUP2 + 3 ROLLD - DUP IF TYPE 0 == THEN
0 R->C SWAP 0 R->C END >>
```

```
'PDIV'                                  Polynomial DIVision
                                        Copyright Hewlett-Packard Co.
<< DUP 1 GET OVER SIZE -> d t n         Used with permission.
                                        See reference 1.
<< { } SWAP DUP SIZE n - 1 + 1 SWAP
START DUP 1 GET t / COLCT ROT OVER 1
->LIST + 3 ROLLD 1 n FOR m OVER m
GET d m GET 3 PICK * - ROT m ROT PUT
SWAP NEXT DROP 2 100 SUB NEXT d >> >>

'NORM'                                  Normalize coefficients
<< PE 1 GET -> n an << PE LIST->
DROP 1 n FOR q an / 'PE' n 1 + q -
ROT PUT NEXT >> >>

'GFIX'                                  Get Display mode
<< 53 54 55 56 1 4 START FS? 4
ROLLD NEXT 1 3 START 2 * + NEXT         Get decimal value of flags
DUP IF 0 == THEN DROP 3 END DUP FIX     FIX if STD mode
NEG ALOG 'ER' STO >>
```

e. References:

1. Mathematical Applications, Hewlett-Packard Co.
   No. 00028-90111, June 1988

2. Numerical Calculations and Algorithms, Beckett & Hurt,
   McGraw-Hill, p. 71.

2. Polynomial Curve Fitting

a. Introduction:

These programs will allow polynomial curve fitting (PCF) to virtually unlimited degree, although a degree higher than 8 is rarely used due to roundoff error.

PCF should be used with caution. For certain functions the values of the fitted polynomial can become unstable and diverge. An example of this is Runge's function: (See reference 1.)

$$y(x) = \frac{1}{1 + 25x^2}$$

The higher the degree of polynomial used, the more unstable the polynomial values near the end points +1 & -1. Hence PCF is not the best interpolation method, although it is satisfactory in many situations. (See Spline Interpolation in section V.4.) It is not intended for extrapolation.

The mathematics used comes from least squares theory (reference 2): Given an array of (x,y) data points we first form an array U containing:

$$U = \begin{vmatrix} 1 & x_1 & x_1^2 & x_1^3 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & x_2^3 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & x_3^3 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 & \dots & x_n^m \end{vmatrix}$$

where n = the number of data points and
m = the polynomial degree and

$m \leq n + 1$.

Then the coefficient array C is given by:

$C = (U^T U)^{-1}(U^T Y)$, where ^{T} indicates matrix transpose and Y is the column vector of y data points $| y_1 \; y_2 \; y_3 \; \dots \; y_n |^T$.

b. Stack diagram:

Level	Before	After
1:	Degree M of polynomial	-

Note: The calculated polynomial coefficients are available for viewing in an array named 'C'. The array 'C' is used for predicted value calculations using the program PRDV.

c. Example:

It is necessary to create the statistics array ΣDAT of x and y data points before using the programs. We create the following x,y data ΣDAT array using the STAT menu functions:

```
[[ 0 20 ]
 [ 1 35 ]
 [ 2 66 ]
 [ 3 119 ]
 [ 4 200 ]
 [ 5 315 ]]
```

[[0 20[1 35[2 66[3 119[4 200[5 315 ENTER Σ+

Since we have six data points we could use a degree as high as five. However we will initially use degree three. The program is started by placing 3 in the command line and pressing PCF.

When the run annunciator goes off, we can try the given integer as well as intermediate values of x to test the goodness of fit.

Put the number 4 in the command line and press PRDV. Level 1 will show (4.00,200.00) in complex format to display (x,y) on one line. (FIX 2). Now try a number beyond the given x inputs. Press 7, PRDV and see (7.00,671). An intermediate value of x = 3.5 gives y = 155.63.

Going to LVL4 subdirectory level, the variable C is the coefficient array and contains:

```
[[ 20.00 ]
 [ 9.00 ]
 [ 5.00 ]
 [ 1.00 ]]
```

The interpolating polynomial is thus: $x^3 + 5x^2 + 9x + 20$.

If we try degree 4 by entering 4, PCF, the array C will contain

```
[[ 20.00 ]
 [ 9.00 ]
 [ 5.00 ]
 [ 1.00 ]
 [ -6.16E-9 ]]
```

Hence degree three is sufficient for these data points.

(Actually, the author "cheated" and formed the ΣDAT array directly from the 3rd degree polynomial given above. But it illustrates the point that a higher degree polynomial is not necessarily a better fit.)

d. Listing:

Program Listing	Comments
`'PCF'`	Main program
`<< LVL4 'M' STO NΣ -> n`	M degree, n no. of data points
`<< n M 1 + 2 ->LIST 0 CON 'U'`	Create U matrix
`STO ΣDAT 1 GCOL`	Get column 1 of ΣDAT
`'XD' STO 'XD' 1 1 n`	Store as XD vector
`FOR i GETI 'X' STO 1 M 1 +`	
`FOR j 'U' i j 2 ->LIST X j 1 - ^`	Fill in U elements
`PUT NEXT NEXT CLEAR U TRN DUP ΣDAT`	
`2 GCOL * SWAP U * / 'C' STO`	Get Y vector & coeff array C
`'X' 'XD' 'U' 3 ->LIST PURGE UP >> >>`	
`'PRDV'`	Predicted value program

```
<< LVL4 -> x << C ARRY-> DROP
1 M START x * + NEXT                    Evaluate using Horner's method
x SWAP R->C UP >> >>                    Display predicted value

'GCOL'                                  Extract column subprogram
<< -> k << TRN ARRY-> 2 GET -> n
<< n NLA -> c2 <<                       Utility subprogram
n NLA c2 IF k 2 == THEN
SWAP END DROP >> >> >> >>

'NLA'                                   Utility subprogram
<< 1 2 ->LIST ->ARRY >>
```

e. References:

1. Computer Methods for Mathematical Computations, Forsythe, Malcolm, & Moler, Prentice-Hall, 1977

2. Spectral Analysis and its applications, Jenkins & Watts, Holden-Day, 1968, p. 132.

3. Surface Fitting

a. Introduction:

Polynomial surface fitting (PSF) is a means of fitting a family of curves simultaneously. For example, transistor Ic vs Vce as a function of Ib represents a family of curves as seen on a curve tracer. The curves can be viewed as projecting out of the page (z-axis) and forming contour lines of a surface. For a transistor, the z-axis would be the base current Ib, the y-axis collector current Ic, and the x-axis the collector to emitter voltage Vce.

Thus instead of merely fitting a polynomial y = f(x) as in the previous section, these programs will fit virtually any number of curves of the form y = f(x,z). To accomplish this we need a list X of the x data points, a list Z of the z data points, and a matrix Y of the y data points as a function of z. That is,

$$X = (x_1\ x_2\ x_3\ \ldots\ x_N),$$

$$Z = (z_1\ z_2\ z_3\ \ldots\ z_K),$$

$$Y = \begin{bmatrix} y_{11} & y_{12} & y_{13} & \ldots & y_{1K} \\ y_{21} & y_{22} & y_{23} & \ldots & y_{2K} \\ y_{31} & y_{32} & y_{33} & \ldots & y_{3K} \\ \vdots & \vdots & \vdots & & \vdots \\ y_{N1} & y_{N2} & y_{N3} & \ldots & y_{NK} \end{bmatrix}$$

where N is the number of x,y data point pairs for each curve and K is the number of curves in the family. (K-1 is the highest power of z in y = f(x,z).).

Another dimension we will need is M, the highest power of x in y = f(x,z), which is a user input.

The general form of y = f(x,z) is

$$\begin{aligned} y = {} & b_{11} + b_{12}z + b_{13}z^2 + \ldots + b_{1K}z^{K-1} \\ & (b_{21} + b_{22}z + b_{23}z^2 + \ldots + b_{2K}z^{K-1})x \\ & (b_{31} + b_{32}z + b_{33}z^2 + \ldots + b_{3K}z^{K-1})x^2 \\ & \quad \vdots \\ & (b_{L1} + b_{L2}z + b_{L3}z^2 + \ldots + b_{LK}z^{K-1})x^M \end{aligned}$$

where L = M+1.

The programs given here create four matrices in the solution process:

Matrix	Dimensions Row	Columns
U	N	M+1
A	K	M+1
ZM	K	K
B	M+1	K

The B matrix contains the coefficients bIJ as given above in the general form of y = f(x,z), and as such, is the solution we are looking for.

The mathematics used is an extension of least squares theory to three dimensions:

$$A = (U^TU)^{-1}(U^TY);\ B = (ZM^{-1}A)^T$$

where ZM is constructed from the z data points in the same manner as U is contructed from the x data points, as was done in the PCF section. The process can be extended to four or higher dimensions (see reference 1).

b. Stack diagram:

Level	Before	After
1:	M	-

Note: As in the PCF section, the B matrix is available for viewing in directory level 4, along with U, A, & ZM. The immediate purpose of the B matrix is to provide Predicted values of Y given X and Z ('PYXZ').

c. Examples:

For the first example, let X = (1 2 3 4), and Z = (0 1 2). Then we have 3 curves in the family, one for z = 0, z = 1, and z = 2. Hence K = 3. Since there are four values in X, N = 4. From given data, the Y matrix is:

```
     [[ 3 14 29 ]
Y =   [ 3 43 91 ]      (dim = N x K)
      [ 3 90 189 ]
      [ 3 155 323 ]]
```

This means that y = f(x,z) = f(1,0) = 3; f(3,2) = 189; f(4,1) = 155, etc.

The lists X and Z plus the Y matrix must be stored in directory level 4 prior to running the program. (The ancillary matrices U, A, and ZM can be purged if so desired in order to optimize the program. They are retained for tutorial purposes in the listings given in d.)

Let M = 2. Then place 2 in the command line and press PSF.

For this example, the calculated matrices should be as follows:

```
     [[ 1 1 1 ]
 U =  [ 1 2 4 ]            [[ 3 0 0 ]          [[ 1 0 0 ]
      [ 1 3 9 ]        A =  [ 3 2 9 ]     ZM =  [ 1 1 1 ]
      [ 1 4 16 ]]           [ 3 8 18 ]]         [ 1 2 4 ]]

     [[ 3 0 0 ]
 B =  [ 0 0 2 ]    (rounded in FIX 0 format)
      [ 0 9 0 ]]
```

Hence we can neatly pick out the nonzero B elements and plug them into

$y = f(x,z) = 3 + 2xz^2 + 9x^2z$

Note that the first row of B gives the z coefficients for x^0, while the second row gives the z coefficients for x^1, and so forth.

Evaluating y = f(x,z) = f(3,2) = 3 + 2*3*4 + 9*9*2 = 3 + 24 + 162 = 189 which agrees with the Y matrix element $y_{x(z+1)} = y_{33}$.

The A matrix provides intermediate answers about each curve in the family. For the A matrix given above we have (for M = 2):

$y_1 = 3 + 0 + 0,$
$y_2 = 3 + 2x + 9x^2$
$y_3 = 3 + 8x + 18x^2.$

In order to get intermediate or predicted values of y for various values of x and z, the PYXZ program is used. Place the value of x in stack level 2 and the value of z in stack level 1, and press PYXZ. The program will return the predicted value of y for those values of x and z. For example, key in x = 2, z = 1. PYXZ should return 42.9999999974 (in STD mode) or close enough to 43 as in the Y matrix. Using PYXZ again for y = f(3.5,2.1) gives y = 265.395.

For a second example, let X = (1 2 3 4), Z = (0 1 2 3); here N = 4, and K = 4. Again from collected data:

```
     [[ 3 27 93 231 ]
Y =   [ 3 64 179 378 ]
      [ 3 119 301 579 ]
      [ 3 192 459 834 ]]
```

Let M = 2 again; place 2 in the command line and press PSF as before.

When the program has finished, the matrices should be:

```
     [[ 1 1 1 ]            [[ 3 0 0 ]            [[ 1 0 0 0 ]
U =   [ 1 2 4 ]        A = [ 8 10 9 ]       ZM = [ 1 1 1 1 ]
      [ 1 3 9 ]             [ 43 32 18 ]          [ 1 2 4 8 ]
      [ 1 4 16 ]]           [ 138 66 27 ]]        [ 1 3 9 27 ]]

     [[ 3 0 0 5 ]
B =   [ 0 4 6 0 ]
      [ 0 9 0 0 ]]
```

As in the first example, we pick out the nonzero b elements and form

$y = f(x,z) = 3 + 5z^3 + 4xz + 6xz^2 + 9x^2z.$

The author has used "textbook" examples here mainly to get the material across. In real world examples, the B coefficients will not be neat little integers. The user must excercise some judgement concerning the relative size of the coefficients choosing only those that will have a predominate affect on the values of y.

d. Listing:

Listing	Comments
'PSF'	Main program
<< LVL4 -> m << X SIZE Z SIZE	Get M, N, & K
-> n k << 1 n FOR i 1 m 1 +	
FOR j 'X' i GET j 1 - ^ NEXT	For matrix U
NEXT n m 1 + 2 ->LIST ->ARRY 'U' STO	
U TRN DUP Y * SWAP U * / TRN 'A'	Solve for A matrix
STO k >> GETZB >> >>	Get ZM & B matrices
'GETZB'	
<< -> k << 1 k FOR i 1 k FOR j	
'Z' i GET j 1 - ^ NEXT NEXT	Form ZM matrix
k k 2 ->LIST ->ARRY 'ZM' STO A ZM	
/ TRN 'B' STO UP >> >>	Solve for B matrix
'PYXZ'	Get y = f(x,z) predicted values
<< LVL4 -> x z << B ARRY-> LIST->	
DROP ( ) -> m k blst << 1 m	Get dimensions of B and start coefficient list
START 1 k 1 - START z * +	Evaluate using Horner's method
NEXT blst + 'blst' STO NEXT	
blst LIST-> DROP 1 m 1 - START	
x * + NEXT >> >> UP >>	

e. References

1. Multidimensional Algebraic Polynomial Approximation, Chandra P. Nehra, IEEE Transactions on Aerospace & Electronic Systems, Vol AES-21, No. 5, Sept, 1985

III. SIGNAL PROCESSING

1. RMS Value of Random Waveforms

a. Introduction:

This program will determine the ac and dc rms value of any random waveform. The number of data points of waveform amplitude that can used is limited only by available memory.

The statistics menu is used to enter the data points into the ΣDAT array. A minimum of 20 points should be used. The more points that are used, the more accurate will be the results.

b. Stack diagram:

Level	Before	After
3:	-	dc rms value
2:	-	average value
1:	-	ac rms value

c. Examples:

For the first example assume the ΣDAT array contains the following data points for a +/- 1V square wave: (Though hardly random, this example illustrates that the program can be used for both deterministic and random waveforms.)

```
1 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

When program 'RRMS' is run, the stack should show (as expected):

```
3:  1.000  (FIX 3 format)
2:  0.000
1:  1.000
```

Next we try a 25% duty cycle rectangular waveform:

1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Running 'RRMS' again results in:

```
3:  0.500
2:  0.250
1:  0.433
```

This example shows that when using an rms voltmeter, one should know if it reads dc or ac rms.

The final example will be truly random, using 20 numbers generated from RAND in the REAL menu:

0.518 0.202 0.715 0.002 0.912 0.306 0.276 0.797 0.562 0.899

0.022 0.079 0.415 0.245 0.965 0.950 0.435 0.621 0.991 0.084

These data points give the following values:

```
3:  0.601
2:  0.500
1:  0.333
```

d. Listing:

Program Listing	Comments
'RRMS'	Random RMS
<< LVL3 VAR NΣ 1 - * NΣ /	ac rms value squared
MEAN SQ -> a d << a d + sqrt	
d sqrt a sqrt UP >> >>	Fill stack

e. References:

The statistical root-mean value, i.e., the squared dc rms value, is given by:

$$V^2rms = \frac{var(n - 1)}{n} + mean^2$$

where var is the variance, mean is the average value, and n is the number of data points. The dc and ac rms values are related by:

$$V^2rms = V^2ac + V^2dc$$

where Vdc is the average or mean value. Hence $V^2ac = var(n - 1)/n$.

2. RMS Value of Deterministic Waveforms

a. Introduction:

If a waveform can be expressed mathematically, then the programs in this section are preferred from a convenience standpoint.

Given a mathematical expression for the waveform, the user must create a subprogram 'FX' to evaluate the expression as a function of time t.

b. Stack diagram:

Level	Before	After
3:	-	dc rms value
2:	-	average value
1:	No. of data points	ac rms value

c. Examples:

The first example will be the absolute value of sin(t) (full-wave rectified sinewave).

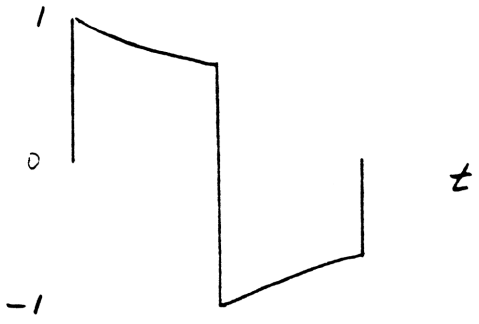
Note: The FX subprogram for this waveform is given below as FX1 so that the subprograms can be separated. As seen from the listing, FX merely calls FX1. This makes it easy to modify FX to call other subprograms with different functions.

When 'FRMS' is executed, the stack shows: (using 20 data points)

```
3:  0.707
2:  0.631
1:  0.318
```

as expected for this waveform. (Use RAD mode.)

Subprogram FX2 gives the differentiated rectangular waveform shown in the figure below:



Using 20 data points again and running 'FRMS' with program FX as << FX2 >> gives the following values:

```
3:  0.815
2:  0.000
1:  0.815
```

d. Listing:

Program Listing	Comments
'FRMS'	
<< LVL3 CLΣ INV -> s <<	step = 1/no. of data points
0 1 s - FOR t t FX Σ+ s	Evaluate the function
STEP RRMS >> >>	When ΣDAT is filled then use RRMS program
'FX'	
<< FX1 >> or << FX2 >>	Select subprogram
'FX1'	
<< 2 pi * * -> t << 'SIN(t)' EVAL	Switch to RAD mode before using.
ABS >> >>	
'FX2'	
<< 1 .5 -> t tc d <<	Time t, time constant tc = 1; duty cycle d = 0.5.
'EXP(-tc*t)'	Positive half cycle 1st.
IF t d ≥ THEN DROP '-EXP(-tc*(t-d))'	Negative half cycle.
END EVAL >> >>	

e. References

See previous section.

3. RMS Value of Piecewise Continuous Waveforms

a. Introduction

This RMS program was developed because many waveforms encountered in electrical engineering are formed from piecewise continuous line segments. Examples are trapezoidal, triangular, sawtooth, and rectangular waveforms.

This program does not make use of the statistics menu in the HP-28; instead, a time vs. amplitude matrix is generated by the user from the given waveform.

b. Stack diagram:

Level	Before	After
3:	-	dc rms value
2:	-	average value
1:	time/amplitude matrix	ac rms value

c. Examples:

The time/amplitude matrix (TAM) general form is as follows:

```
[[ t1   V1 ]      (t1 is usually zero)
 [ t2   V2 ]
   .    .
   .    .
   .    .
 [ tn   Vn ]]     (n limited only by memory)
```

The first example will be the +/- 1V square wave described by 20 discrete data points in III.1.:

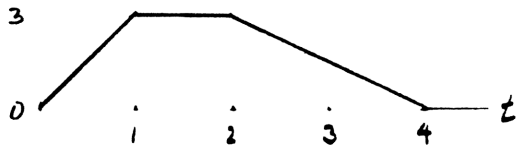
```
[[ 0 0 ]
 [.0001 1 ]     (100 micro-second rise time to 1V)
 [ .9999 1 ]
 [ 1.0001 -1 ]  (200 micro-second fall time to -1V)
 [ 1.9999 -1 ]
 [ 2 0 ]         (100 micro-second rise time to 0V )
```

Placing this matrix in level 1 and executing 'LRMS' gives:

```
3:  1.000      (FIX 3 format)
2:  0.000
1:  1.000
```

The next TAM describes the waveform shown next to it:

```
[[ 0 0 ]
 [ 1 3 ]
 [ 2 3 ]
 [ 4 0 ]]
```



Running 'LRMS' with this matrix in level 1 gives:

```
3:  2.121
2:  1.875
1:  0.992
```

d. Listing:

Program Listing	Comments
'LRMS'	
<< TRN ARRY-> LIST-> DROP SWAP DROP -> m	Get no. of points
<< m ->LIST DEPTH	
ROLLD m ->LIST 0 0 -> v t s1 s2	Get time & amplitude lists & zero summation variables.
<< 1 m 1 - FOR i t i GET	
t i 1 + GET v i GET v i 1 + GET	Get pairs of time & amplitude in variables
-> t1 t2 v1 v2 <<	t1 t2 v1 v2
t2 t1 - v1 DUP * v1 v2 * + v2 DUP * + *	(t2-t1)(v1v1+v1v2+v2v2)
s1 + 's1' STO	Sum in s1 for rms
t2 t1 - v1 v2 + * s2 + 's2' STO	(t2-t1)(v1+v2), sum in s2 for dc average
IF i m 1 - == THEN t2 END >> NEXT -> t2	When done get period T = t2
<< s2 t2 2 * /	
DUP SQ s1 t2 3 * / sqrt DUP SQ SWAP	Fill stack to level 3
4 ROLLD SWAP -	
sqrt 3 FIX >> >> >> >>	

e. Reference:

The (V^2rms) value can be obtained by integrating $(f(t))^2$ over the limits from 0 to period T, and then multiplying by 1/T to get the average. It can be shown by operating on successive piecewise continuous line segments by this procedure that the squared dc rms value is given by:

$$V^2rms = (1/3T) \sum_{i=1}^{L} (t_{i+1} - t_i) f(V_i, V_{i+1})$$

where $f(V_i, V_{i+1}) = V_iV_i + V_iV_{i+1} + V_{i+1}V_{i+1}$, and L = no. of line segments.

By integrating the line segments to get the average value Vdc:

$$Vdc = (1/2T) \sum_{i=1}^{L} (t_{i+1} - t_i)(V_i + V_{i+1})$$

For example, using the second example TAM where L = 3 and T = 4:

```
[[ 0 0 ]
 [ 1 3 ]
 [ 2 3 ]
 [ 4 0 ]]
```

$Vdc = (1/8)[(1-0)(3+0) + (2-1)(3+3) + (4-2)(3+0)]$

$= (1/8)[\ 3 + 6 + 6\ ] = 15/8 = 1.875.$

$V^2rms = (1/12)[(1-0)(0+0+9) + (2-1)(9+9+9) + (4-2)(9+0+0)]$

$= 54/12.$

$Vrms = sqrt(54/12) = 2.121.$

$Vac = sqrt(54/12 - (15/8)^2) = 0.992.$

4. Discrete Fourier Transform

a. Introduction:

The Discrete Fourier Transform (DFT) provides the harmonic amplitude and phase of a digitized waveform. The purpose of these programs is to provide the harmonic amplitudes only, since the phase angles of the harmonics are usually of lesser interest. The phase is computed though and is available for scrutiny if desired.

By using N data points, the first N/2 harmonics of the waveform can be obtained. A value of 40 for N is usually satisfactory, since harmonics beyond the 20th are seldom of interest.

(The Fast Fourier Transform (FFT) is given in the next section. The FFT becomes valuable when N is very large, e.g., N = 2048. All else being equal, the FFT requires more code to implement than does the DFT and requires more memory for the larger complex data points. The price paid for using the DFT instead of the FFT is of course longer execution time.)

b. Stack diagram:

Level	Before	After
1:	N	Harmonic plot

c. Examples:

The user must provide a vector X of the N data points. This can be done manually point-by-point, or programmatically using a subprogram to generate the data points. In the example below, subprogram VIN3 generates the following vector X of 20 data points:

X = [1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

This of course represents a 1V, 0.25 duty cycle rectangular waveform. Placing 20 in level one and pressing 'DFT' will start the program.

The program automatically jumps to a harmonic plot program called 'FPLT' that will plot the M harmonics. The computed harmonics are contained in a vector C1 as follows:

C1 = [0.452 0.324 0.156 0.000 0.100 0.124 0.079 0.000 0.072 0.100]

This is the familiar 2Ad|sinx/x| harmonic plot for rectangular pulses where x = dMpi, d = the duty cycle factor (0.25 for the example above), M = the harmonic number, A is the amplitude, and pi = 3.141..

According to the program, the 3rd harmonic should be close to 0.156:

dMpi = (0.25)(3)(3.14159) = 2.356; (2)(0.25)|sin(2.356)/(2.356)| = 0.150;

The first harmonic should be close to 0.452:

dMpi = 0.785; 0.5|sin(0.785)/0.785)| = 0.450.

Hence the harmonics calculated by the program are approximately correct. When the duty factor is changed to 0.5, the harmonics are:

C1 = [0.639 0.000 0.220 0.000 0.141 0.000 0.112 0.000 0.101 0.000]

As expected, the even harmonics are zero, as every 1/d harmonic is for rectangular pulses.

c. Listing:

Program Listing	Comments
'DFT'	
<< RAD LVL3 DUP 2 / -> n m <<	Get N and M.
n 1 ->LIST 0 CON 'X' STO	Omit this line if manually creating the X vector in level 3
m 1 ->LIST	
0 CON DUP 'C1' STO 'P' STO	P is for harmonic phase angle storage.
n VIN	Omit this line if manually creating the X vector
1 m FOR j 0 0 R->C	Start outer harmonic loop
'C' STO 1 n FOR i i j 2 pi	Start inner time loop
* * * -> a3 << C RE a3 n / COS 'X' i	
GET -> te << te * + C IM a3 n / SIN	te is temporary storage
te * + R->C 'C' STO >> >> NEXT	Next time point
'C1' j C ABS 2 * n / DUP	Multiply harmonics by 2/n (2/T for continuous FT)
IF 0.0001 < THEN DROP 0 END	Force zero harmonics
PUT 'P' j C ARG NEG PUT NEXT	Next harmonic
'C' PURGE UP 23 MENU m >>	Housekeeping
FPLT >>	Jump to harmonic plot
'FPLT'	
<< LVL3 -> m << 0 0 R->C DUP PMIN AXES	
m 0.5 R->C PMAX CLLCD DRAX 1 m FOR j	The 0.5 may have to be increased or decreased for better resolution
'C1' j GET j SWAP R->C PIXEL NEXT UP >> >>	

```
'VIN'  << VIN3 >>
'VIN3'  << .25 -> n k                          Store n & duty cycle factor
<< 'X' 1 1 ->LIST 1 n FOR i
IF k i n / ≥ THEN 1                          Amplitude = 1
ELSE 0 END PUTI NEXT CLEAR >> >>               Amplitude = 0
```

d. References:

1. Advanced Engineering Mathematics, E. Kreyszig, 2nd Ed., Wiley, p. 458.

2. Topics in Advanced Mathematics for Electronics Technology, S. Paull, 1966, Wiley, p. 63.

5. Fast Fourier Transform

a. Introduction:

This famous algorithm, developed by Cooley and Tukey in 1965, greatly improves the speed of the DFT computation. As stated previously, its use justifies the longer program length when using large arrays of input data.

Some significant differences that must be observed when using the FFT program given here, is that the input data must be complex, and the number of complex data points N must be an integer power of two, such as 32, 256, etc. Also, the spectrum will repeat itself after N/2 harmonics due to aliasing. The FFT program provides N harmonics for N inputs, but only the first N/2 harmonics have significance.

b. Stack diagram

For program 'FFT':

Level	Before	After
1:	-	-

For program 'FPLT':

1:	Maximum harmonic amplitude	Harmonic plot

c. Example

Using a 25% duty cycle rectangular waveform again, the following vector X of input complex data points must be created: (N = 16)

```
X = [ (1,0) (1,0) (1,0) (1,0) (0,0) (0,0) (0,0) (0,0)

      (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) ]
```

When this vector is stored, the program can be run by pressing FFT. A program called GABS given in the Listing section generates the absolute value or magnitude of the complex output harmonics which are stored in rectangular form in the same X vector.

After running GABS, the stack will show: (FIX 3)

```
16:  0.500  (16 harmonics for 16 inputs)
15:  0.453
14:  0.327
13:  0.159
12:  0.000  (every fourth or 1/.25 harmonic is zero)
11:  0.106
10:  0.135
9:   0.090
8:   0.000  (aliasing starts)
7:   0.090
6:   0.135
5:   0.106
4:   0.000
3:   0.159
2:   0.327
1:   0.453
```

Other than the top harmonic (2 x dc average), the stack levels designate the harmonic number. This is due to the odd symmetry of the spectrum (folded about N/2).

Note: Both the input data points and the output harmonics are stored in the same vector 'X' to save memory. This is known as in-place computation. After determining the maximum harmonic amplitude, the FPLT program (not the same as FPLT as described in the DFT section) can be used to plot the harmonics.

Note that the amplitudes are approximately the same as those in the DFT section, since the input waveform is the same.

d. Listing.

Program Listing — Comments

```
'GABS'

<< LVL3 X ARRY-> LIST-> DROP -> n

<< 1 n START ABS 2 * n / n ROLLD NEXT >> UP >>
```

```
'FFT'
<< LVL3 X SIZE LIST-> DROP 'N' STO          Store no. of data points
N LOG 2 LOG / .5 + IP 'V' STO               N = 2^V
0 'M' STO CS S2 S5 S11                      Start algorithm. See ref-
                                            erence for description.
                                            S2 & S5 refer to Step 2
                                            & Step 5, etc. of algorithm.
0 10000 START IF P N S - 1 - == THEN        Program will be stopped by 'S18'
S5 END S11 NEXT UP >>

'CS'                                        Calculate S
<< 2 V M - 1 - ^ 'S' STO >>

'S2'                                        Step 2 of algorithm
<< 0 'P' STO BFC                            BFC is ButterFly Computa-
                                            tion
DO 'P' 1 STO+ BFC UNTIL P S 1 - == END >>   Increment P and repeat BFC

'S5'                                        Step 5
<< 1 'M' STO+ CS                            Increment M and recalculate S
IF M V 1 - == THEN S18 ELSE S2 END >>

'S11'                                       Step 11
<< P S 1 + + 'P' STO BFC 0 'K' STO          Start auxiliary counter K
DO 'P' 1 STO+ 'K' 1 STO+ BFC UNTIL
K S 1 - == END >>
```

```
'S18'                                   Step 18
<< 0 'P' STO BFC DO 'P' 2 STO+ BFC      Last stage of algorithm
UNTIL P N 2 - == END
( M P Q S K V ) PURGE BREV              Bit REVerse routine
UP KILL >>                              Stop program.

'BFC'                                   ButterFly Computation
<< P S + 'Q' STO P S MOD 2 M ^ * -> r   Get exponent r
<< XPQ + XPQ - 0 pi 2 * N / r * NEG
R->C EXP * 'X' Q 1 + ROT PUT 'X' P 1 +  Store in-place
ROT PUT >> >>

'XPQ'                                   Utility routine
<< 'X' P 1 + GET 'X' Q 1 + GET >>       Get previous values

'BREV'                                  Bit REVerse routine
<< 1 -> j << 1 N 1 - FOR i IF i j <
THEN 'X' j GET -> t << 'X' i GET 'X' j
ROT PUT t 'X' i ROT PUT >> END N 2 /
-> k << WHILE k j < REPEAT j k - 'j' STO
k 2 / 'k' STO END j k + 'j' STO >> NEXT >> >>

'FPLT'                                  Frequency PLoT
<< LVL3 0 0 R->C PMIN N SWAP            Max amplitude in stack prior
                                        to running
R->C PMAX (0,0) AXES CLLCD DRAX 'X'
2 1 N 1 - FOR i GETI ABS 2 * N / i SWAP ABS => magnitude
R->C PIXEL NEXT CLEAR UP >>
```

e. Reference

1. Digital Signal Processing, Oppenheim & Shafer, Prentice-Hall, 1975

Additional reference:

Most textbooks provide FFT flow diagrams for the case of N = 8, and leave as a problem of how to extrapolate the algorithm for N = 16, 32, etc. The author has done this for the general case and offers the following step-by-step procedure upon which the above programs are based.

The decimation-in-frequency butterfly computation is defined as (see reference 1.)

$$X_{M+1}(P) = X_M(P) + X_M(Q) \qquad (1)$$

$$X_{M+1}(Q) = [X_M(P) - X_M(Q)]W_N^R \qquad (2)$$

where $W_N = \exp(-2j\pi/N)$ ($\pi = 3.141...$)

Since N must be an integer power of 2, let $N = 2^V$, and the stage counter M will range from 0 to V - 1. Hence for N = 16, the maximum value of M is 3.

For the variables P & Q, we must first define an auxiliary variable

$$S = 2^{V-M-1}$$

The variable P will start from zero be incremented in a manner to be described below and

$$Q = P + S$$

One more variable is the exponent R of the complex operator W_N. The variable R is given by

$$R = 2^M \text{MOD}(P,S)$$

where MOD is the modulus or remainder function.

Now all that is needed is N (which is given), M and P, and everything else (V, S, Q & R) follows. Every time that P is incremented, the butterfly computations (1) and (2) must be performed. The results are stored in-place for later use when the stage counter M is incremented and to conserve memory.

The general procedure for incrementing P and M and consequently performing the FFT is as follows:

Step	Operation	Do (1) & (2) ?
1.	Set P = M = 0	Yes
2.	P -> P + 1	Yes
3.	If P = S - 1 go to step 5.	No
4.	Go to step 2.	No
5.	M -> M + 1	No
6.	If M = V - 1 go to step 18.	No
7.	P = 0	Yes
8.	P -> P + 1	Yes
9.	If P = S - 1 go to step 11.	No
10.	Go to step 8.	No
11.	P -> P + S + 1	Yes
12.	K = 0 (Auxillary counter)	No
13.	P -> P + 1, K - > K + 1	Yes
14.	If K = S - 1 go to step 16.	No
15.	Go to step 13.	No
16.	If P = N - S - 1 go to step 5.	No
17.	Go to step 11.	No
18.	P = 0	Yes
19.	P = P + 2	Yes
20.	If P = N - S - 1 STOP	No
21.	Go to step 19.	No

The variables should increment as shown below for N = 16:

M = 0; S = 8			M = 1; S = 4			M = 2; S = 2			M = 3; S = 1		
P	Q	R	P	Q	R	P	Q	R	P	Q	R
0	8	0	0	4	0	0	2	0	0	1	0
1	9	1	1	5	2	1	3	4	2	3	0
2	10	2	2	6	4	4	6	0	4	5	0
3	11	3	3	7	6	5	7	4	6	7	0
4	12	4	8	12	0	8	10	0	8	9	0
5	13	5	9	13	2	9	11	4	10	11	0
6	14	6	10	14	4	12	14	0	12	13	0
7	15	7	11	15	6	13	15	4	14	15	0

The algorithm just described is a decimation-in-frequency FFT in which the final output is in "bit-reversed" order. The BREV routine given above in affect converts P & Q to binary numbers, reverses the sequence of 1's and 0's so obtained, and converts back to decimal form. This will arrange the output harmonics in sequential order. The BREV routine was translated from a FORTRAN sequence given in reference 1.

6. Discrete Convolution

a. What it does:

Discrete time convolution, as opposed to continuous time convolution, convolves samples of continuous time waveforms at discrete amplitudes.

The algorithm can be very easily explained by comparing the procedure to multiplying two polynomials together. For example, we multiply the two polynomials:

$$Pa(s)Pb(s) = (1 + 2s + 3s^2 + 4s^3)(5 + 4s + 3s^2 + 2s^3 + s^4)$$

$$= (5 + 4s + 3s^2 + 2s^3 + s^4) + (10s + 8s^2 + 6s^3 + 4s^4 + 2s^5)$$

$$+ (15s^2 + 12s^3 + 9s^4 + 6s^5 + 3s^6) + (20s^3 + 16s^4 + 12s^5 + 8s^6 + 4s^7)$$

To collect like powers of s we can group them in columns and add as follows:

			columns				
s^0	s^1	s^2	s^3	s^4	s^5	s^6	s^7
5	4	3	2	1			
	10	8	6	4	2		
		15	12	9	6	3	
			20	16	12	8	4
5	14	26	40	30	20	11	4

The product of the two polynomials is thus:

$$5 + 14s + 26s^2 + 40s^3 + 30s^4 + 20s^5 + 11s^6 + 4s^7.$$

We have just convolved two discrete time waveforms which can be represented as vectors: A = [1 2 3 4] and B = [5 4 3 2 1]

or A*B = B*A = [5 14 26 40 30 20 11 4], where the "*" means convolution.

b. Stack diagram:

Level	Before	After
2:	vector A (or B)	-
1:	vector B (or A)	C = A*B = B*A

c. Examples

Using some examples from reference 1.:

A = [1 0.8 0.64 0.512 0.4096 0.32768 0.262144 0.2097152]

(Note: $A = 0.8^N$, N = 0, 1, 2, ... ,7)

B = [1 1 1 1 1 1 1 1] (Discrete unit step)

Then placing these vectors in the stack and pressing 'VCNV' should show vector C in level 1 as:

C = [1 1.8 2.44 2.952 3.362 3.688 3.951 4.16 3.16 2.36 1.72 1.208

0.799 0.472 0.21]

(Reference 1, pp 14, 15)

The number of elements in C is Na + Nb - 1, where Na, & Nb are the number of elements in vectors A & B.

The next example shows that a vector can be time-shifted by a shifted unit-sample pulse:

A = [0 0 1 0 0]

B = [5 4 3 2 1]

Pressing 'VCNV' with A and B in the stack (in either order):

C = [0 0 5 4 3 2 1 0 0]

or vector B time-shifted two units to the right with 5 + 5 - 1 = 9 elements.

c. Listing

Listing	Comments
'VCNV'	Vector CoNVolution
<< LVL4 -> a b << a SIZE LIST-> DROP	Get dimensions of A & B
b SIZE LIST-> DROP -> n m << m n + 1 -	Na + Nb - 1
1 ->LIST 0 CON -> c <<	Create storage space for C
1 n FOR i 1 m FOR j a i GET b j GET	Get A & B elements
* c i j + 1 - -> ij << ij GET + c	Add corresponding columns
ij ROT PUT 'c' STO >> NEXT NEXT c UP	Put C in stack
>> >> >> >>	

References:

1. Digital Signal Processing, Oppenheim & Schafer, Prentice-Hall, 1975

7. Autocorrelation

a. Introduction:

The autocorrelation function relates the statistical dependence, or correlation, of neighboring values in a time series. Thus if a time series is made up of perfectly random noise, then each time point is statistically independent of the previous time point, and the autocorrelation is zero or random itself.

Related to autocorrelation is cross correlation, where dependence between two separate time series is measured.

It is important to note at the outset that the algorithm used in this program is an estimate of discrete time autocorrelation and is used primarily in statistical applications. Deterministic or analytical autocorrelation uses an integration method similar to convolution.

The expression used here is

$$Cov(k) = \frac{1}{N} \sum_{i=1}^{N-k} (X(i) - xb)(X(i+k) - xb)$$

$$k = 0,1,2,\ldots, N-1$$

where Cov(k) is the covariance (sometimes called autocovariance) of the kth term, xb is the mean of the data, X(i) is the ith data point, X(i+k) is k time points away from X(i), and N is the number of data points.

To get the autocorrelation, the above summation is normalized by dividing by the mean square value of the data or

$$Cor(k) = Cov(k)/Cov(0)$$

The user must create a set of data points using the statistics menu, thus creating a ΣDAT array. This can be done manually or using a function subprogram to generate the data points and create the ΣDAT array. The dimension need only be N x 1, where N is the number of data points. That is, it is not necessary to use a time vs. amplitude (N x 2) array.

b. Stack diagram:

For 'ACFCN':

Level	Before	After
1:	-	Autocorrelation vector

For 'APLT'

2:	Maximum plot ordinate	-
1:	Minimum plot ordinate	Plot

c. Examples:

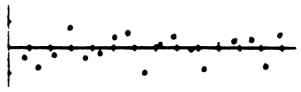
Assume the ΣDAT array is made up of the following 20 data points using the RAND function (RNDed in FIX 3 display mode):

.604, .033, .889, .363, .246, .230, .872, .817, .372, .414,
.622, .105, .641, .971, .896, .414, .452, .570, .924, .003

After running the 'ACFCN' program, level 1 contains the following vector (named 'CKA') with N-1 or 19 points:

[-.159, -.249, -.101, .235, -.150, -.079, .105, .189, -.347, .055,
.154, -.035, -.270, .050, .068, .112, -.213, .159, -.024]

Plotting this vector using 'APLT' gives the plot below, with ,5 and -,5 as the maximum and minimum ordinates.



The plot shows that the random number generator is almost truly random, i.e., white noise, as the correlation bounces up and down with no discernable pattern.

d. Listing

Listing	Comments
'ACFCN'	AutoCorrelation FunCtioN
<< LVL3 N MEAN -> n xb << n 1 ->LIST	
0 CON 'CKA' STO 0 n 1 - FOR k 1 n k -	Begin outer loop
FOR i ' DAT' i GET xb - ' DAT' i k +	Begin inner loop
GET xb - * k CK1P GET + k CK1P ROT PUT	CK1P, utility routine
NEXT k CK1P DUP2 GET n / PUT NEXT >>	Multiply by 1/N
CKA ARRY-> LIST-> DROP ROLL -> ck0	Get Cov(0)
<< DEPTH 1 ->LIST ->ARRY ck0 / 'CKA'	Normalize, get Cor(k)
STO CKA UP >> >>	
'CK1P'	Utility routine
<< -> k << 'CKA' k 1 + >> >>	
'APLT'	Plot program
<< LVL3 CKA SIZE LIST-> DROP -> n	Get N
<< 0 SWAP R->C PMIN n SWAP R->C PMAX	
(0,0) AXES CLLCD DRAX 'CKA' 1 1 ->LIST	
1 n 1 - FOR i GETI i SWAP R->C PIXEL	
NEXT CLEAR UP >> >>	

e. References

1. Spectral Analysis and its Applications, Jenkins & Watts, Holden-Day, 1968, p.180

IV. TRANSFER FUNCTIONS

1. Step Response of Any Transfer Function
(Distinct roots)

a. Introduction:

The step response of an input-output transfer function is a measure of stability, or lack of it. If an output increases in value, or oscillates at a constant amplitude, the system is unstable. If the ringing is damped and decays to zero, then the system is marginally stable. If the output overshoots one or two times before settling, then the system bandwidth, gain, and stability are probably just about right, depending on the application.

The usual method of determining a step response, is to do a partial fraction expansion of the transfer function (including the 1/s step input), after the roots of the denominator characteristic equation have been found. Then it is a relatively simple matter to determine the term-by-term exponential response, either real or complex, to obtain the output as a function of time.

A simple example is:

$$F(s) = \frac{24}{s(s+2)(s+3)(s+4)} = \frac{A}{s} + \frac{B}{s+2} + \frac{C}{s+3} + \frac{D}{s+4}$$

$$A = F(s)s\Big|_{s=0} = \frac{24}{24} = 1; \quad B = F(s)(s+2)\Big|_{s=-2} = \frac{24}{(-2)(3-2)(4-2)} = -6;$$

$$C = F(s)(s+3)\Big|_{s=-3} = 8; \quad D = F(s)(s+4)\Big|_{s=-4} = -3.$$

For complex roots the procedure is the same.

In using the following programs, it is necessary to know the roots of the denominator characteristic equation. The roots can be obtained from the root-finding programs in section II. 1., Real or Complex Roots, n > 4.

Prior to running the program, the stack must contain a list of numerator polynomial coefficients and a list of denominator roots. For the example above, the numerator list would be (24) and (2 3 4) for the denominator roots.

(The s = 0 for the 1/s step input is not included. Also note that the roots are as they appear in the denominator, not -2, -3, & -4 which are the actual signs of the roots.)

Once the partial fraction expansion coefficients have been determined, the next step is to get the time response. A second program 'GTR' (for Get Time Response) requires the stack to contain the maximum time and the time step before executing. The user may have to experiment with these two parameters before the time plot program 'TPLT' is run. The TPLT program requires the maximum and minimum plot amplitudes in the stack prior to running. These can be obtained by examining the stack time response and choosing the amplitudes that will give the best resolution.

b. Stack diagram:

For 'SRTF' (Step Response Transfer Function)

Level	Before	After
2:	(numerator coeff)	-
1:	(denominator roots)	No displayed output

For 'GTR' (Get Time Response)

Level	Before	After
k: (= Tmax/Tstep + 1)	-	Time response at T(k) = Tmax
k-1:	-	Time response at T(k-1)
.	.	.
.	.	.
.	.	.
2:	Tmax	Time response at T(1) = Tstep
1:	Tstep	Time response at T(0) = 0

After examining the results of the time response in the stack, then run TPLT.

For 'TPLT' (Time PLoT)

Level	Before	After
2:	Max amplitude	-
1:	Min amplitude	Time plot

c. Examples:

Again using the same example given in a. above, the numerator and denominator lists are placed in the stack as follows:

```
2:  { 24 }
1:  { 2 3 4 }
```

Then the starting program SRTF is run, which provides no output to the stack. The user then makes an estimate of the maximum time and the time step of the time response. Given the roots -2, -3, & -4, a maximum time of 1 second, with a time step of 0.1 second is chosen:

```
2:  1.0
1:  0.1
```

Running GTR gives the following stack in (time,amplitude) FIX 2 format:

```
11:  (1.00,0.53)
10:  (0.90,0.46)
 9:  (0.80,0.39)
 8:  (0.70,0.32)
 7:  (0.60,0.24)
 6:  (0.50,0.17)
 5:  (0.40,0.11)
 4:  (0.30,0.06)
 3:  (0.20,0.02)
 2:  (0.10,3.20E-3)
 1:  (0.00,0.00)
```

As a check, we know the explicit time response to be:

$f(t) = 1 - 6e^{-2t} + 8e^{-3t} - 3e^{-4t}$, which at t = 1 is

$f(1) = 1 - 6(0.135) + 8(0.050) - 3(0.018)$

$= 1 - 0.812 + 0.398 - 0.055 = 0.531$ <------- checks

To plot this time response, a max amplitude of 0.6 and min of 0 should fill up the LCD fairly well. CLEAR the screen of the time response and enter in the stack:

```
2:  0.60
1:  0.00  (The minimum amplitude will be zero most of the time.  But
           there will be ringing responses that will go negative.)
```

The following plot was obtained by using a time step of 0.05 to get more plotting points:

Next we try something more interesting; complex roots with some ringing responses:

$$\text{Let } F(s) = \frac{5}{s[(s+1)^2 + 2^2]}$$

Key in the numerator coefficient list and denominator root list as

```
2:  ( 5 )
1:  ( (1,2) (1,-2) )
```

and press SRTF. Guessing that 1.0 second will show the complete response, enter:

```
2:  1.0
1:  0.1
```

and press GTR. Stack level 11: shows (1.00,0.99). To insure a settled response, CLEAR the stack and try:

```
2:  4.0
1:  0.2
```

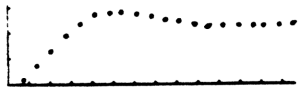
and press GTR. The stack should now show: (FIX 3)

```
21:  (4.000,0.994)        10:  (1.800,1.185)
20:  (3.800,0.984)         9:  (1.600,1.207) (Peak overshoot)
19:  (3.600,0.973)         8:  (1.400,1.191)
18:  (3.400,0.963)         7:  (1.200,1.120)
17:  (3.200,0.957)         6:  (1.000,0.986)
16:  (3.000,0.959)         5:  (0.800,0.789)
15:  (2.800,0.972)         4:  (0.600,0.545)
14:  (2.600,0.998)         3:  (0.400,0.293)
13:  (2.400,1.037)         2:  (0.200,0.086)
12:  (2.200,1.087)         1:  (0.000,0.000)
11:  (2.000,1.140)
```

Hence a plot amplitude of 1.3 should work fine. CLEAR and enter

```
2:  1.300
1:  0.000
```

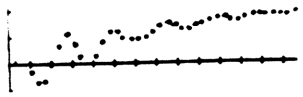
and press TPLT, which gives the plot shown below:



As a grand finale, we examine the following transfer function:

$$F(s) = \frac{200s^2 - 3280s + 6560}{s(s + 0.5)[(s + 1)^2 + 9^2][(s + 2)^2 + 12^2]}$$

Enter (200 -3280 6560) in level 2: and (.5 (1,9) (1,-9) (2,12) (2,-12)) in level 1 and run SRTF. Choosing a max time of 4.0 seconds, a 0.1 second time step, a maximum and minimum amplitude of 1.0 and -0.5, we get the following plot after running GTR and TPLT in turn:



Note that the negative dip of the response is caused by the -3280 term in the numerator.

d. Listing:

Program Listing	Comments
'SRTF'	
<< LVL4 'DLST' STO 'NLST' STO	
GAAT	Get A matrix (See reference)
0 N FOR I I GNUM	Get numerator. N is denominator order.
1 'D9' STO	Initialize denominator of partial fraction expansion
1 N FOR J 'A' I 1 + J 2 ->LIST GET	
'D9' STO* NEXT N9 D9 /	N9 = num. magnitude
'KA' I 1 + ROT PUT NEXT	Put coeff in KA array
'N9' PURGE 'D9' PURGE UP >>	Housekeeping
'GAAT'	Get A minus A Transpose
<< RAD NLST SIZE 1 - 'M' STO	Get numerator order
DLST SIZE 'N' STO N 1 + N 2	
->LIST (0,0) CON 'A' STO	Create N+1 by N A array
N 1 + 1 ->LIST (0,0) CON 'KA' STO	Storage for residues
0 N FOR I 0 N FOR J IF J 0 == THEN 0	
ELSE 'DLST' J GET END NEXT NEXT N 1 +	
DUP 2 ->LIST ->ARRY DUP TRN CONJ - GA >>	

```
'GA'                                          Get A array
<< -> za << 1 N 1 + FOR I 1 CF 1 N
FOR J IF 1 FC? THEN IF I J == THEN 1          Skip main diagonal
                                               and form A
SF END END 'A' za I J IF 1 FS? THEN
1 + END 2 ->LIST GET I J 2 ->LIST SWAP
PUT NEXT NEXT >> >>                           Return to SRTF

'GNUM'                                        Get NUMerator magnitude
<< -> I << NLST LIST-> DROP FLIP              Reverse list order
IF M 0 == THEN 'N9' STO ELSE IF I 0           If M=0 then done
== THEN 0 ELSE 'A' 1 I 2 ->LIST GET NEG
END -> x << 1 M START x * + NEXT 'N9' STO     Evaluate using Horner's
                                               method
>> END >> >>                                  Return to SRTF

'FLIP'                                        Reverse list order
<< 1 M 1 + FOR q q ROLL NEXT >>

'TPLT'                                        Time PLoT
<< LVL4 0 SWAP R->C PMIN ET SWAP R->C PMAX
(0,0) AXES CLLCD DRAX 'TRL' ET DT / 1 ->LIST
GETI DROP 0 ET START GETI PIXEL DT STEP
DROP2 UP >>
```

```
'GTR'                                          Get Time Response
<< LVL3 'DT' STO 'ET' STO                      Store time step & max time
1 CF { } 'TRL' STO                             Begin Time Response List
0 ET FOR t KA ARRY-> LIST-> -                  Get residue
DROPN C->R DROP 'M1' STO                       Get Re(residue)
1 N FOR i 'A' 1 i 2 ->LIST GET DUP IM
IF 0 == THEN i t REAL                          Get real coeff
ELSE i t CMPX END                              or complex coeff
NEXT t M1 R->C TRL + 'TRL' STO                 Add to Time Response List
DT STEP TRL LIST-> DROP 'M1' PURGE UP >>       Done

'REAL'                                         Real coeff evaluation
<< -> i t << C->R DROP NEG t * EXP
'KA' i 1 + GET C->R DROP * 'M1' STO+ >> >>

'CMPX'                                         Complex coeff evaluation
<< -> i t << IF 1 FS? THEN CLEAR 1 CF
ELSE 1 SF C->R SWAP NEG t * EXP 'KA' i 1 +
GET DUP ABS 2 * ROT * ROT NEG t * ROT ARG +
COS * 'M1' STO+ END >> >>                      e^-at cos(Bt + arg)
```

e. Reference:

In finding the constants (residues) associated with each partial fraction term, a pattern forms that is amenable for computer programming as shown below using an N = 4 example:

$$\text{Let } F(s) = \frac{N(s)}{s(s + p1)(s + p2)(s + p3)(s + p4)},$$

where N(s) is a polynomial in s, the p's are roots (real or complex poles) of the denominator equation, and the 1/s input is included. The procedure is:

$$K0 = F(s)s\Big|_{s=0} = \frac{N(0)}{p1p2p3p4},$$

$$K1 = F(s)(s + p1)\Big|_{s=-p1} = \frac{N(-p1)}{-p1(p2 - p1)(p3 - p1)(p4 - p1)}$$

$$K2 = F(s)(s + p2)\Big|_{s=-p2} = \frac{N(-p2)}{-p2(p1 - p2)(p3 - p2)(p4 - p2)}$$

Continuing this for K3 and K4, we can form an N+1 x N+1 matrix of the five denominators as follows:

$$ZA = \begin{vmatrix} 0 & p1 & p2 & p3 & p4 \\ -p1 & 0 & p2 - p1 & p3 - p1 & p4 - p1 \\ -p2 & p1 - p2 & 0 & p3 - p2 & p4 - p2 \\ -p3 & p1 - p3 & p2 - p3 & 0 & p4 - p3 \\ -p4 & p1 - p4 & p2 - p4 & p3 - p4 & 0 \end{vmatrix}$$

To create ZA an auxiliary matrix A1 is first formed:

$$A1 = \begin{vmatrix} 0 & p1 & p2 & p3 & p4 \\ 0 & p1 & p2 & p3 & p4 \\ 0 & p1 & p2 & p3 & p4 \\ 0 & p1 & p2 & p3 & p4 \\ 0 & p1 & p2 & p3 & p4 \end{vmatrix}$$

ZA is then given by ZA = A1 - $A1^T$. After eliminating the main diagonal we get an N+1 x N matrix A:

$$A = \begin{vmatrix} p1 & p2 & p3 & p4 \\ -p1 & p2 - p1 & p3 - p1 & p4 - p1 \\ -p2 & p1 - p2 & p3 - p2 & p4 - p2 \\ -p3 & p1 - p3 & p2 - p3 & p4 - p3 \\ -p4 & p1 - p4 & p2 - p4 & p3 - p4 \end{vmatrix}$$

What's the point of all this? To collect in one location the factors of the denominators of K0 thru K4. By multiplying across each row of A we get the value of variable D9 in the program SRTF for each K0 thru K4. Since the numerator is a polynomial, it is easy to evaluate it and store as variable N9 in program GNUM using Horner's method.

It might help if we use the first example where NLST = (24) and DLST = (2 3 4) and walk through the various arrays and variables:

The program GAAT creates the ZA matrix from the A1 matrix and its transpose, and it will be for this example:

$$ZA = \begin{vmatrix} 0 & 2 & 3 & 4 \\ 0 & 2 & 3 & 4 \\ 0 & 2 & 3 & 4 \\ 0 & 2 & 3 & 4 \end{vmatrix} - \begin{vmatrix} 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{vmatrix} = \begin{vmatrix} 0 & 2 & 3 & 4 \\ -2 & 0 & 1 & 2 \\ -3 & -1 & 0 & 1 \\ -4 & -2 & -1 & 0 \end{vmatrix}$$

With the main diagonal of zeros eliminated by program GA we have the complex A matrix, which for this example the elements are all real:

$$A = \begin{vmatrix} (2,0) & (3,0) & (4,0) \\ (-2,0) & (1,0) & (2,0) \\ (-3,0) & (-1,0) & (1,0) \\ (-4,0) & (-2,0) & (-1,0) \end{vmatrix}$$

The program GTR decides which subroutine to go to, REAL or CMPX, based on the imaginary parts of A elements. If zero, then the program branches to REAL, if not zero then to CMPX.

The vector KA, containing the residues of the partial fraction expansion, is formed by SRTF and is:

KA = | K0 K1 K2 K3 |

= | (1,0) (-6,0) (8,0) (-3,0) |

where each element is formed by the ratio N9/D9. For example, multiplying the 2nd row of matrix A gives D9 = -4. When this is divided into the numerator magnitude N9 = 24, the 2nd element of KA = -6. The subroutine REAL performs the evaluation

$$f(t) = 1 - 6e^{-2t} + 8e^{-3t} - 3e^{-4t}$$

and sums into variable M1 for each time point. When the summation is done the final value of M1 is stuffed into the TRL list for that time point.

For a transfer function with complex roots, the procedure is the same except for the f(t) evaluation. For example let

$$F(s) = \frac{N0}{s[(s + a)^2 + b^2]}$$

Then

$$f(t) = 1 + 2|K1|e^{-at}\cos(bt - \tan^{-1}a/b)$$

where

$$|K1| = \frac{(a^2 + b^2)^{1/2}}{2b}$$

To use a specific example, let a = 1 and b = 2, then K1 = $(5)^{1/2}/4$ = 0.559, and $\tan^{-1}$-0.5 = -2.678 radians (in the 4th quadrant). If (5) and ((1,2) (1,-2)) is placed in the stack and SRTF run, the polar form of KA vector will be:

KA = [K0 K1 K2]

= [(1,0) (0.559,-2.678) (0.559,2.678)]

Note that K1 and K2 are conjugates.

2. Bode Plots From Transfer Functions

a. Introduction:

These programs are relatively straight-forward useage of the HP-28S functions to plot a frequency response (Bode) plot directly from a transfer function F(s) = F(jw)

b. Stack diagram:

The stack is not used as an input. The MENU command is used to input three parameters for the plot: beginning log frequency (BF), points per decade (PD), and number of decades (ND). When the program has finished the stack will contain:

Level	After
1:	("LogF dBV Deg") for each frequency

For the plot function 'BPLT'

Level	Before	After
2:	Max amplitude	-
1:	Min amplitude	Bode plot

To plot phase instead of magnitude, set flag 2.

c. Example:

Assume we desire the frequency response of the following biquad transfer function:

$$F(s) = \frac{s^2 + N1s + N0}{s^2 + D1s + D0}$$

The constants have been determined from a bandpass filter with a center frequency of 50 KHz to be:

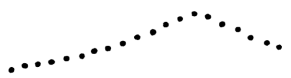
N1 = 1483709; N0 = 100250626566

D1 = 150376; D0 = N0.

These constants must be manually stored in LVL4 subdirectory prior to running the program. Then let BF = 4 (10 KHz), PD = 20, & ND =1 will provide the frequency response from 10 KHz to 100 KHz (1 decade) using 20 equally spaced points of log frequency. The output list in level 1: can be edited to display:

```
1:  ( "4.00 2.830 38."   10 KHz
      "4.05 3.381 41."
      "4.10 4.012 44."
      "4.15 4.728 47."
      "4.20 5.535 49."
      "4.25 6.441 51."
      "4.30 7.455 53."
      "4.35 8.590 54."
      "4.40 9.867 55."
      "4.45 11.311 54."
      "4.50 12.957 52."
      "4.55 14.833 48."
      "4.60 16.910 39."
      "4.65 18.895 24."
      "4.70 19.881 1."   Note sudden phase shift at about 50 KHz.
      "4.75 19.050 -22."  (Log 50,000 = 4.699)
      "4.80 17.110 -38."
      "4.85 15.023 -47."
      "4.90 13.124 -52."
      "4.95 11.458 -54."
      "5.00 9.996 -55." )  100 KHz
```

To plot this put 25 in level 2: and 0 in level 1:, which produces the plot shown below:



d. Listing:

Program Listing	Comments
'TFBP	Transfer Function Bode Plot
<< LVL4 (STO BF PD ND) MENU HALT	Get plot parameters
BF ND + 'FL' STO CLEAR BF FL	FL = last frequency
FOR f f 'F' STO TFCN	Get specific transfer function
R->P C->R SWAP LOG 20 * F 2 FIX	Set up display strings
->STR " " + SWAP 3 FIX RND ->STR + " "	
+ SWAP 0 FIX RND ->STR + PD	
INV STEP DEPTH ->LIST 'VOUT' STO VOUT	Store output in VOUT list
3 FIX UP 23 MENU >>	
'TFCN'	Specific transfer function
<< F ALOG 2 pi * * 0 SWAP R->C -> s	s = jw
'(s^2+N1*s+N0)/(s^2+D1*s+D0)' EVAL >>	
'BPLT'	Bode PLoT
<< LVL4 BF SWAP R->C PMIN FL SWAP R->C	Setup Pmin and Pmax
PMAX BF 0 R->C AXES CLLCD DRAX 'VOUT'	
1 1 ->LIST BF FL START GETI STR->	
IF 2 FS? THEN SWAP END DROP	To plot phase, set flag 2
R->C PIXEL PD INV STEP DROP2 UP >>	

e. References:

1. Principles of Active Network Synthesis & Design, G. Daryanani, Wiley, 1976.

V. MISCELLANEOUS

1. Worst Case Analysis

a. Introduction:

There are in general three types of worst-case circuit or function analysis: Extreme Value Analysis (EVA), Root-Sum-Square (RSS), and Monte Carlo Analysis (MCA). The three methods can best be explained by a simple example.

Suppose the function to be analyzed is $F = \dfrac{A1\ M1}{1 + \dfrac{R1 + R2}{R3}}$,

Let the component values and tolerances be as follows:

Component	Value	Tolerance ± %
A1	2	2
M1	1	1
R1	5	0.5
R2	4	0.5
R3	1	0.5

The nominal value of F is 2/(1 + 9) = 0.2. To find the maximum or high value of F, the procedure is to vary each of the components in the same direction, say +1%, one at a time, and note the change in the value of F, whether it increases or decreases, and record the sign, + or -. This gives us the sign of the partial derivative of F with respect that component. Before multiplying the next component by 1.01 (+1%), we change the previous component back to its nominal value.

(In more complicated functions one has to be careful, as it is possible for the partial sign to change from say a +0.5% perturbation to a +1% purturbation. That is, the sign may be negative at +0.5%, but be positive at +1% purturbation. Fortunately this is a rare circumstance.)

Continuing with our EVA of F, we construct a table for the changes in F with +1% changes in all the components, one at a time. Let F′ be the value of F with each of the component changes:

Component	F	F′	sgn(F′ - F)	Comments
A1 + 1%	0.2	0.202	+	
M1 + 1%	0.2	0.202	+	A1 back to 2.0
R1 + 1%	0.2	0.1990	-	M1 back to 1.0, etc.
R2 + 1%	0.2	0.1992	-	
R3 + 1%	0.2	0.2018	+	

Then the EVA high value of F will be with A1 + 2%, M1 + 1%, R1 - 0.5%, R2 - 0.5%, and R3 + 0.5%. Substituting these values into F, F becomes 0.2079. Reversing the tolerance sign on these components gives us the EVA low value of F: F = 0.1923. Note that the high and low value is not symmetrical.

Assuming Gaussian or normal distribution for the component statistics, the chances of F attaining either the high or low value is extremely remote. Increase the parts count and it becomes even more remote. Nevertheless, in some military and space applications, EVA is required as part of the contract. (MIL-STD-785.)

A more realistic worst case analysis used in commercial and some military applications is RSS. The mathematics for RSS is given in the Reference section. For now the basic idea is the assumption of Gaussian statistics for the components, and then calculating the square root of the sum of the $(F - F')^2$ values. This gives a Gaussian distribution for F with the magnitude of the standard deviations as part of the calculation. Then depending on the ground rules, either one, two or three standard deviations (sigmas) can be taken at the worst case values.

The + and - 3 sigma values for F are +3s = 0.2046, -3s = 0.1954. Note that this spread is less than the EVA for F. This will usually, but not always, be the case.

The last method is MCA, which is in general equivalent to the RSS method. In this method, a random number is generated and is used to determine the random value of A1 within ± 2%. A second random number is then used to find a random value for M1 within + 1%, and so forth through the five components. The (random) value of F is then calculated using these first five random values for the components. This is repeated a sufficient number of times to get a spread for the values of F. No program is given for MCA, as essentially the same information can be obtained from RSS.

b. Stack diagram

EVA

Level	Before	After
3:	-	EVA high value
2:	-	Nominal value
1:	-	EVA low value

RSS

3:	-	+3s value
2:	-	Nominal value
1:	-	-3s value

c. Examples:

Using the five component example above as the first example, the user must create and store the following algebraic objects:

Object	Name stored under
'T(5)'	'M1'
'2*T(4)	'A1'
'T(3)'	'R3'
'4*T(2)'	'R2'
'5*T(1)'	'R1'

Store the following under the name of FUNC:

```
<< 'A1*M1/(1+(R1+R2)/R3)' EVAL >> 'FUNC' STO
```

Finally a vector of decimal tolerances named 'D' must be created to follow the order of the T subscripts given above as follows:

```
[ .5 .5 .5 2 1 ], 'D' STO
```

That is, T(1) thru T(3) are 0.5%, T(4) is 2%, and T(5) is 1%.

Then EVA may be pressed to give the following output: (FIX 4)

```
3:  0.2079
2:  0.2000
1:  0.1923
```

Clear stack and press RSS for the following output:

```
3:  0.2046
2:  0.2000
1:  0.1954
```

The worst case programs can also be used for functions that vary with time or frequency by embedding RSS and EVA in a time or frequency loop. To demonstrate this, change the FUNC program to:

```
<< 0 2 pi F * * R->C -> s << '(s^2+N1*s+N0)/(s^2+D1*s+D0)'

EVAL ABS >> >>
```

We will now proceed to calculate EVA and RSS values for this bi-quadratric (AKA biquad) as a function of frequency. Store the following algebraics under the names shown:

Object	Name stored under
'1483709*T(1)'	'N1'
'100250626566*T(2)'	'N0'
'150376*T(3)'	'D1'
'100250626566*T(4)'	'D0'

Assuming 5% tolerances for all four variables, vector D will be

```
[ 5 5 5 5 ] 'D' STO
```

We will be using a main program that iteratively calls either RSS or EVA (not both) depending on the status of flag 1. If flag 1 is set, then RSS is called; if clear then EVA. This new main program is named FRWR. Prior to executing FRWR, the stack must contain the following logarithmic frequency parameters:

```
3:  Log beginning frequency  (e.g., log 100 Hz = 2, etc.)
2:  Points per decade
1:  Number of decades
```

The biquad with the given coefficients is a bandpass filter with a center frequency of 50 KHz. Thus we should look at the frequency range of from 10 KHz to 100 KHz with 10 points per the one decade. The stack should then contain:

```
3:  4  (log 10,000 = 4)
2:  10 (points per decade)
1:  1  (1 decade)
```

Running FRWR gives us the following outputs for EVA: (Flag 1 clear and flag 2 set to get dBV)

```
[[ 4     3.727 2.83  1.934 ]   (i.e, EVA high is 3.727 dBV)
 [ 4.1   4.922 4.012 3.104 ]    & EVA low is 1.934 dBV at 10 KHz.)
 [ 4.2   6.466 5.535 4.608 ]
 [ 4.3   8.418 7.455 6.498 ]
 [ 4.4   10.888 9.867 8.86  ]
 [ 4.5   14.079 12.957 11.861 ]
 [ 4.6   18.166 16.91  15.69  ]
 [ 4.7   20.72  19.881 18.95  ]  (Note peaking at 10^4.7 =
 [ 4.8   18.148 17.11  16.087 ]   50119 Hz)
 [ 4.9   13.87  13.124 12.369 ]
 [ 5     10.571 9.996 9.405 ]]
```

Running FRWR with flag 1 set will give a matrix of values for RSS, with the spread at each frequency less than the corresponding EVA run. By definition of $\pm$ 3 sigma, the spread about the nominal will be symmetrical. Not so for EVA though.

c. Listing

Program Listing	Comments

Note: EVA and RSS can be used standalone for dc (single-valued) functions as shown in the first example. FRWR below is for functions of frequency only.

```
'FRWR'
<< -> bf pd nd << 3 FIX bf DUP nd +          Start frequency loop
FOR q q ALOG 'F' STO IF 1 FS? THEN RSS       Flag 1 set => RSS
ELSE EVA END q 4 ROLLD IF 2 FS? THEN GDB     Use dBV if flag 2 set
END pd INV STEP 'F' PURGE CTA 23 MENU >> >>

'GDB'                                        Get DB
<< 1 3 START LOG 20 * 3 ROLLD NEXT >>

'RSS'                                        Root Sum Square
<< LVL3 LVL4 0 'S' STO 1.0001 -> p           Perturbation factor
<< FRMT 1 NC FOR i 'T' i p PUT V0 FUNC       Evaluate function
- 'D' i GET 100 / * SQ 'S' STO+ 'T' i 1      Restore tolerance
                                             multiplier to 1
PUT NEXT S sqrt p 1 - / DUP V0 + SWAP
V0 SWAP - V0 SWAP UP >> >>                   Setup stack for +3s, nom,
                                             & -3s.

'EVA'
<< LVL3 LVL4 FRMT 1 NC FOR i 'T' i 'D'
i GET 100 / 1 + PUT FUNC 'VOUT' i ROT PUT
'T' i 1 PUT NEXT 3 CF WCHL 3 SF WCHL
V0 SWAP UP
```

```
'WCHL'                                          Worst Case High Low

<< 1 NC FOR I 'T' I 'VOUT' I GET V0

- SIGN IF 3 FS? THEN NEG END 'D' I GET

100 / * 1 + PUT NEXT FUNC >>
```

```
'FRMT'                                          FoRM T

<< 35 36 CF CF D SIZE LIST-> DROP 'NC' STO      Store no. of components NC

NC 1 ->LIST DUP 1 CON 'T' STO 0 CON             Set up T and VOUT vector

'VOUT' STO FUNC 'V0' STO >>                     V0 = nominal value
```

```
'CTA'                                           Convert To Array

<< DEPTH 4 / 4 2 ->LIST ->ARRY RND STD >>
```

e. References:

The EVA technique is described in the first example. For RSS, the following development is used:

Using the first example function, the variance of the output is:

$$\sigma_o^2 = \left(\frac{\partial V_{o1}}{\partial R_1}\cdot\sigma_1\right)^2 + \left(\frac{\partial V_{o2}}{\partial R_2}\cdot\sigma_2\right)^2 + \left(\frac{\partial V_{o3}}{\partial R_3}\cdot\sigma_3\right)^2 + \left(\frac{\partial V_{o4}}{\partial A_1}\cdot\sigma_4\right)^2 + \left(\frac{\partial V_{o5}}{\partial m_1}\cdot\sigma_5\right)^2$$

Using R1 as an example, the partial derivatives are approximated as

$$\frac{\partial V_{o1}}{\partial R_1} \cong \frac{\Delta V_{o1}}{\Delta R_1} = \frac{\Delta V_{o1}}{PR_1}$$

where P = 0.0001. Assuming normal (Gaussian) distribution, the component standard deviations (sigmas) are approximated as

$$\sigma_1 \cong \frac{D_1 R_1}{3}$$

where Di is the decimal tolerance. Then the standard deviation of the output is

$$\sigma_o = \frac{1}{3P}\left[(\Delta V_{o1} D_1)^2 + (\Delta V_{o2} D_2)^2 + \cdots + (\Delta V_{o5} D_5)^2\right]^{1/2}$$

2. Accurate Gain Ratios

a. Introduction

In choosing the resistor values for an inverting opamp with a gain of -R2/R1 for a given required gain G, the usual procedure is to choose a convenient value for R1 such as 10K, and then solve for R2 by calculating R2 = R1 x G. The next step is to find the closest standard value to the calculated value for R2 from standard resistor value tables.

Depending on the difference between the calculated and standard value of R2, a gain error is introduced. Intuitively one knows that among the 48, 96, or 192 standard resistor values there is a best combination of R1 and R2 that will be closer to the design gain G. This program finds that best combination.

It will find the best combination for three circuit configurations shown below with the corresponding limits on gain G:

Inverting opamp: G = -R2/R1 (-100 < G < -0.01)

Non-inverting opamp: G = 1 + R2/R1 (1 < G < 100)

Voltage divider: G = 1/(1 + R1/R2) (0.01 < G < 0.99)

The user can choose from 48 2% values, 96 1% values, or 192 0.5% values (same as 0.1% values). Appropriate decade values such as 1.92K, 19.2K, 192K, etc. must be provided by the user.

b. Stack diagram

Level	Before	After
3:	-	G
2:	% tolerance, 2, 1, or 0.5 *	R2
1:	G	R1

* For 0.1% 192 values, use 0.5%.

Press 'GRM' to start the program.

c. Examples:

Three examples are given in tabular form below for 1% values:

Configuration	Desired Gain G	R1	R2	Error
Voltage divider	0.24	4.75	1.50	0
Inverting opamp	-0.68	1.50	1.02	0
Non-inverting opamp	1.70	1.50	1.05	0

The error is the actual gain vs. the desired gain. Of course, even with 192 values, the error will not always be zero, but it will be the smallest possible value.

c. Listing

Program Listing	Comments
'GRM'	Main program.
<< LVL3 2 FIX 'G' STO 96 SWAP / 'B' STO	B = 48, 96, or 192
5 CF G IF G 0 < THEN INV NEG RER END	Choose gain path
IF G 1 < G 0 > AND THEN INV 1 - RER END	
IF G 2 < G 1 > AND THEN 1 - INV RER END	
IF G 2 > THEN 1 - 5 SF RER END UP	
'RER'	REsistor Ratio
<< 'A' STO B 'E' STO IF A 1 < THEN A	See reference 1.
INV 'A' STO 5 SF END A LOG B * 0 FIX RND	
1 - 'L' STO 1 B FOR m m L + 'N' STO N B /	
ALOG IF N B ≤ THEN 2 FIX RND END	
IF N B > N B 2 * ≤ AND THEN 1 FIX RND END	
IF N B 2 * > THEN 0 FIX RND END 'R8' STO m	
1 - B / ALOG 2 FIX RND 'R9' STO R8 R9 / A	R8 & R9 are trial values
- ABS 'E1' STO IF E1 E < THEN E1 'E' STO	
R8 'R1' STO R9 'R2' STO END NEXT	
IF 5 FS? THEN R1 R2 'R1' STO 'R2' STO END	
G R2 R1 >>	Put G, R2, & R1 on stack.

e. Reference:

1. Standard Resistors Give Accurate Ratios, R. Boyd, Electronic Design, Sept 8, 1988, p. 111.

3. Spline Interpolation

a. Introduction:

Cubic spline interpolation is one of the better if not the best of the interpolation methods. When one runs into trouble with a polynomial curve fit (PCF) that is unstable at the end points, the spline method will provide much improved performance over PCF.

b. Stack diagram

Level	Before	After
2:	-	u
1:	u, Value to be interpolated.	y = f(u)

c. Examples:

The first example will be the infamous Runge's Function that gives PCF such a difficult time near the end points. The following vectors 'X' and 'Y', representing the x and y values of (for this example) Runge's Function must be created and stored:

[-.6 -.4 -.2 -.1 0 .1 .2 .4 .6] 'X' STO

[.1 .2 .5 .8 1 .8 .5 .2 .1] 'Y' STO

(The number of elements in 'X' and 'Y' must be equal.)

Runge's function is

$$y = f(x) = \frac{1}{1 + 25x^2}$$

To get an interpolated value , say 0.25, place 0.25 in the stack (u), and press CUSPL. The output should be:

```
2:  0.250 (the value of u repeated back)
1:  0.387 ( y = f(u) )
```

For u = -0.58:

```
2: -0.580
1:  0.109
```

The user can create a program to create 'X' and 'Y' instead of keying the vectors in manually. The program CUSPL can also be easily modified to iterate u to create multiple interpolations of some function. The statistics menu can be used to input the x,y data points thus creating the ΣDAT array. A subroutine must then be written to separate the x and y columns into the 'X' and 'Y' vectors.

For a second example, assume the 'X' and 'Y' vectors are:

X = [0 .25 .375 .5 .75]

Y = [13.6 8.9 6.7 5.3 4]

The following values of u and y = f(u) should be obtained:

u	f(u)
0.05	12.782
0.10	11.869
0.15	10.895
0.20	9.894
0.25	8.900

d. Listing

Program Listing	Comments
'CUSPL'	Main program
<< LVL4 X SIZE LIST-> DROP -> u n << n 1	This series of programs were translated from a FORTRAN program in reference 1. See reference 1 for details.
->LIST 0 CON DUP DUP 'B' STO 'C' STO	
'D' STO n SPLINE n u SEVAL u SWAP >> UP >>	
'SPLINE'	
<< -> n << IF n 3 < THEN SPLN4 ELSE 'X'	
2 GET 'X' 1 GET - 'D' 1 ROT PUT 'Y' 2 GET	
'Y' 1 GET - 'D' 1 GET / 'C' 2 ROT PUT 2 n 1 -	
FOR i 'X' i 1 + GET 'X' i GET - DUP DUP 'D' i	
ROT PUT 'D' i 1 - GET + 2 * 'B' i ROT PUT	
'Y' i 1 + GET 'Y' i GET - SWAP / DUP 'C'	
i 1 + ROT PUT 'C' i GET - 'C' i ROT PUT NEXT	
'D' 1 GET NEG 'B' 1 ROT PUT 'D' n 1 - GET NEG	
'B' n ROT PUT 'C' 1 0 PUT 'C' n 0 PUT n 3	
IF ≠ THEN n SPLN1 END n SPLN2 n SPLN3 END >> >>	

```
'SPLN1'
<< -> n << 'C' 3 GET 'X' 4 GET 'X' 2 GET - /
'C' 2 GET 'X' 3 GET 'X' 1 GET - / - 'C'
1 ROT PUT 'C' n 1 - GET 'X' n GET 'X' n
2 - GET - / 'C' n 2 - GET 'X' n 1 - GET
'X' n 3  - GET - / - 'C' n ROT PUT 'C' 1
GET 'D' 1 GET SQ * 'X' 4 GET 'X' 1 GET
- / 'C' 1 ROT PUT 'C' n GET NEG 'D' n
1 - GET SQ * 'X' n GET 'X' n 3 - GET -
/ 'C' n ROT PUT >> >>

'SPLN2'
<< -> n << 2 n FOR i 'D' i 1 -
GET 'B' i 1 - GET / -> t << 'B' i GET t
'D' i 1 - GET * - 'B' i ROT PUT 'C' i
GET t 'C' i 1 - GET * - 'C' i ROT PUT >>
NEXT 'C' n GET 'B' n GET / 'C' n
ROT PUT 1 n 1 - FOR j n j - -> i <<
'C' i GET 'D' i GET 'C' i 1 +
GET * - 'B' i GET / 'C' i ROT PUT >>
NEXT 'Y' n GET 'Y' n 1 - GET - 'D'
n 1 - GET / 'C' n 1 - GET 'C' n GET 2 *
+ 'D' n 1 - GET * + 'B' n ROT PUT >> >>
```

```
'SPLN3'
<< -> n << 1 n 1 - FOR i 'Y' i 1 +
GET 'Y' i GET - 'D' i GET / 'C' i 1 +
GET 'C' i GET 2 * + 'D' i GET * - 'B' i
ROT PUT 'C' i 1 + GET 'C' i GET - 'D'
i GET / 'D' i ROT PUT 'C' i GET 3 *
'C' i ROT PUT NEXT 'C' n GET 3
* 'C' n ROT PUT 'D' n 1 - GET 'D' n ROT
PUT >> >>

'SPLN4'
<< 'Y' 2 GET 'Y' 1 GET  - 'X' 2 GET 'X'
1 GET - / DUP 'B' 1 ROT PUT 'B' 2 ROT PUT >>

'SEVAL'                                          Spline EVALuation
<< -> n u << u 'X' 1 GET IF < THEN n u
SEV1 ELSE u 'X' 2 GET IF < THEN 1 u
SEV2 ELSE n u SEV1 END END >> >>

'SEV1'
<< SWAP DUP 1 + 1 -> u n j i << DO i j + 2
/ -> k << u 'X' k GET IF < THEN k 'j' ELSE
k 'i' END STO >> UNTIL j i 1 + ≤ END i u
SEV2 >> >>
```

```
'SEV2'

<< -> | u << u 'X' | GET - -> d1 << 'Y'

| GET d1 'B' | GET d1 'C' | GET d1 'D' |

GET * + * + * + >> >> >>
```

e. References.

1. Computer Methods for Mathematical Computations, Forsythe, Malcom, & Moler, Prentice-Hall, 1977, p.76

2. Numerical Methods for Scientists and Engineers, R. W. Hamming, 2nd Ed., 1973, p.349.

4. Kalman Filtering

a. Introduction:

The Kalman Filter (KF) is a sophisticated method of extracting a signal that is buried in noise. It can find a signal where there appears to be none. The mathematics combines state variable and statistical methods for multiple-input multiple-output systems such as inertial navigation systems.

The program given here is a first order (scalar) KF. First order means there is one signal source and one output. The HP-28S is certainly capable of implementing a multi-dimensional KF and the reader is encouraged to pursue the subject further to this end. (See reference.) Hence the program presented here is more tutorial than practical.

b. Stack diagram

Level	Before	After
KFIL		
1:	N, number of points	(Input-Output List)
KPLT		
2:	Maximum ordinate	
1:	Minimum ordinate	Input-Output Plot

c. Example.

One example is given similar to that given in reference 1. During program execution, two numbers will appear in the display: The Kalman gain k, and the error covariance p. Both numbers should converge to a steady state value.

The noisy input and filtered output are overlaid on the same plot to show the signal versus the noise. Each run will be slightly different due to the random nature of the input. The output is essentially the same for every run, which demonstrates the essence of the KF.

The input is random "white noise" applied to a first order exponential filter with a time constant of 0.8. Hence the input is filtered white noise.

For an input of (N =) 20 points, the Kalman gain in the display should settle to about 0.167, while the error covariance should converge to about 0.200. The user can EDIT the input-output list in the stack at program completion to determine the minimum and maximum ordinate values for the plot.

For a run of 20 points, put 20 in the stack (or command line) and press KFIL. The input-output list for this run was (again, each run will be slightly different.):

```
( (.352,1.947)   t = 20  (complex form:  (input,output))
  (.549,1.927)   t = 19
  (.564,1.872)   t = 18
  (.146,1.814)   t = 17
        .
        .
        .

  (          )   t = 2
  (          )   t = 1
```

For the plot, place 2 in level 2 and 0 in level 1 and press KPLT, which produced the following plot.

d. Listing

```
'KFIL'
<< LVL3 'N' STO ( ) 'KFO' STO 0.98                  Create storage and
                                                    initialize
1 .04 1 0 1 -> a h q r x p <<
RZG 1 N FOR i 'a*p*a+q' EVAL                        In a multi-dimensional
                                                    KF, the lc variables
'p' STO 'p*h*INV(h*p*h+r)' EVAL                     would be arrays.
-> k << CLLCD k 1 DISP  p 2 DISP
'(1-k*h)*p' EVAL 'p' STO 'a*x'
EVAL 'x' STO 'Z' i GET -> zv <<                     Z is random variable
                                                    vector
'x+k*zv' EVAL 'x' STO zv x                          zv is input,
                                                    x is output.
R->C RND KFO + 'KFO' STO >> >>                      Add to output list
NEXT CLMF KFO UP >> >>                              Put output list on
                                                    stack.
'RZG'                                               Random Z Generator
<< N 1 ->LIST 0 CON 'Z' STO
1 N FOR i 'Z' i i RFG PUT NEXT >>

'RFG'                                               Random Function
                                                    Generator
<< -> i << RAND 2 sqrt * i N /
.8 * NEG EXP * >> >>

'KPLT'                                              Kalman PLoT
<< LVL3 0 SWAP R->C PMIN N SWAP R->C PMAX           Set vertical limits
(0,0) AXES CLLCD DRAX 1 CF PLT                      Input plot
1 SF PLT                                            Output plot
CLEAR UP >>
```

```
'PLT'                                          PLoT routine

<< 1 N FOR I 'KFO' N I - 1 + GET I

SWAP C->R IF 1 FS? THEN SWAP END

DROP R->C PIXEL NEXT >>
```

e. Reference.

1. Introduction to Random Signal Analysis and Kalman Filtering, R. G. Brown, Wiley, 1983

(Of the many books and papers written about Kalman filtering, this is by far the best one to start with.)

5. Newton's Method

a. Introduction:

This program does essentially the same thing as the ROOT function in the HP-28S, but in addition will find complex roots. To find the complex roots of polynomials, the programs given in section I.4., are recommended. The program given here will find real or complex roots for many other functions.

b. Stack diagram

Level	Before	After
1:	Initial (real or complex) guess	Solution

c. Examples

The user must create a function subprogram ('FX') that a solution is being sought for. These functions are generally of the type form x = f(x), where all the x terms cannot be brought over to the left hand side. A simple example is x = ln(x).

The first function of this type to be solved for is

ln(x) + 3x = 11, or ln(x) + 3x - 11 = 0.

Placing an initial guess of 1.0 in level 1, and pressing NM gives (after about 5 seconds) a value of 3.272. That this is indeed a root can be shown by substitution back into the function.

(The initial guess cannot be zero.)

To see if there is any complex roots to this function, we place (1,1) in level 1, and very quickly the answer is

(3.272, -3.029E-14)

which should be interpreted as a real root. Hence there are no complex roots for the given function.

Second example: Find two roots of $x^5 + 32 = 0$.

Placing (1,1) in level 1 as the initial guess, the program comes back with (1.618,1.176). This is the same answer obtained from 32, CHS, ENTER, .2, ^, i.e., without using the program. The textbook answer is 2(cos36 + jsin36). However, this is the only root this method will find. Using a guess of (-1,2), the program will find a second root of 2(cos108 + jsin108) = (-0.618,1.902).

There are 3 other roots equally spaced around a circle of radius 2 in the complex plane. That is, 2(cos324 + jsin324) is also a root.

d. Listing

Program Listing — Comments

```
'NM'

<< LVL4 .0001 -> d << DUP d * 'DX' STO

DO 'X0' STO X0 DUP DUP DX + FX SWAP FX /                    F(X0)
                                          X1 = X0 + -----
1 - DX SWAP / - DUP d * 'DX' STO 'X1' STO X1                F'(X0)

UNTIL X1 X0 - ABS d < END ( X0 X1 DX ) PURGE UP >> >>


'FX'  (1st example)

<< -> x << x LN x 3 * + 11 - >> >>


'FX'  (2nd example)

<< -> x << x 5 ^ 32 + >> >>
```

e. Reference.

1. HP-25 Applications Programs, Hewlett-Packard Co. Rev E 7/76. p. 76

2. HP-35 Math Pac, Hewlett-Packard Co. & Lee Skinner, p. 67.

6. Inductor Design

a. Introduction

One of the obstacles in becoming familiar with the general subject of magnetics, is the sometimes confusing array of units. To name a few: Teslas, Gauss, Webers, Maxwells, Amp-turns, oersteds, Gilberts, Amps per centimeters squared, circular-mils per Amp, Amps per circular-mil (!), Amps per square inch, etc. Fortunately, the HP-28S CONVERT function is a big help.

In an attempt at consistency, the mks (SI) system of units will be used. Then flux density is in Teslas, (symbol B), and field intensity is in Ampere-turns (symbol H). For current density, the consistency battle is almost lost. The various conversions are given below:

(The abbreviation c.m. will be used for circular mils to distinguish it from cm (centimeters)).

$$\frac{4E6 \text{ c.m.}}{\pi \text{ in}^2} = 1, \ \pi = 3.141... \qquad (1)$$

$$\frac{4E6 \text{ c.m.}}{\pi(2.54)^2 \text{ cm}^2} = 1 \qquad (2)$$

$$\frac{4E10 \text{ c.m.}}{\pi(2.54)^2 \text{ m}^2} = 1 \qquad (3)$$

Example: Convert 2mA/c.m. to A/in². Use (1):

$$\frac{0.002 \text{ A}}{\text{c.m.}} \times \frac{4E6}{\pi} \frac{\text{c.m.}}{\text{in}^2} = \frac{8E3 \text{ A}}{\pi \text{ in}^2} = 2546 \text{ A/in}^2$$

Some wire gauge equations used in the programs are:

$$AWG = -20\log((\pi)D),$$

$A = D^2 \times 1E6$. Then substituting:

$$AWG = 60 - 20\log((\pi)A^{1/2}),$$

where D is wire diameter in inches, A is wire area in c.m., and AWG is the wire gauge number.

For the first example below, the program output will be the area-product, abbreviated Ap. This number is in units of meters^4 and is used to select a core. It is the product of window area Wa and core cross-sectional area Ac. Most core catalogs do not give this number directly, but do list Wa in c.m. and Ac in cm^2. The Wa in c.m. must be converted to cm^2 using (2) above and then Ap will be units of cm^4, which is fairly common. The conversion from cm^4 to m^4 is easily done by multiplying Ap in cm^4 by 1E-8.

Some core catalogs do not give Wa and provide only Ac in cm^2 and Ap in units of (c.m.)(cm^2). To get around some of the units confusion, the author has annotated his core catalogs for Ap in m^4 using the following short programs:

```
     Place Ac in cm^2 and Wa in c.m. in stack levels 1: and 2:
     respectively and execute 'AP';

'AP'

<< LVL3 1 CMDM2 INV "m^2" "cm^2" CONVERT DROP * * "cm^4" "m^4"

CONVERT UP >>

'CMDM2'

<< "m^2" "mil^2" CONVERT DROP 4 * pi / >>
```

For example, the Magnetics, Inc. MPP303 Power Core catalog for core # 55206: Wa =225600 c.m.; Ac = 0.226 cm^2. Executing AP gives an Ap of 2.583E-9 m^4. Some users may prefer to drop the last conversion in AP and leave Ap in cm^4.

b. Stack diagram:

Level	Before	After
6:	-	Core area product Ap
5:	-	No. of turns
4:	-	Wire gauge to use.
3:	-	Copper loss in watts
2:	-	Air gap in meters.
1	-	mH/1000T (A_L)

Note: The inductor design program 'COIL', is self-prompting. No stack inputs are required.

c. Examples:

The inputs are: Inductor dc current = I = 10A
Wire current density = cmA = 208 c.m./A
Inductance = L = 37.5 uH
Max flux density = Bm = 0.27 T (Teslas)
Window utilization factor = Ku = 0.3 (dimensionless)
Resistivity of copper = Rhocu = 1.7241E-8 ohm-m
Permeability of free space = uo = 1.2566E-6 Henry/m

These inputs must be stored in LVL3 subdirectory as follows:

Value	Stored name
10	'I'
208	'cmA'
37.5E-6	'L'
0.27	'Bm'
0.30	'Ku'
1.7241E-8	'Rhocu'
1.2566E-7	'uo'

In the inductor design, these parameters are usually fixed, and only the prompted variables of core area (Ac) and mean length of turn (MLT) will be changed during the calculations.

Press 'COIL' to start the program. The first output will be:

"Ap(m^4) = 4.879E-9".

A search of Magnetics, Inc. ferrite core catalog (Ref 2.) shows a 2616 pot core with an Ap of 0.078E6 (c.m.)(cm^2) = 3.952E-9 m^4. The usual practice is to select a core with a higher Ap than that calculated. However, we will see what this solution gives us.

This core has the following values needed to continue the inductor design:

Ac = 0.94 cm^2 and

Mean length of turn = MLT = 0.173 ft = 5.2 cm.

Pressing CONT displays the prompt Ac(cm^2) ?. Key in 0.94 and press CONT. The next prompt will be MLT(cm) ?. Key in 5.2 and press CONT. In a few seconds, the stack will show:

```
6:  "Ap(m^4) = 4.879E-9"
5:  "N = 15."
4:  "AWG = 17."
3:  "Pcu(W) = 1.257"
2:  "Gap = 6.877E-4"
1:  "mH/1000T = 172.
```

If a core catalog shows the air gap in inches, the conversion is easy. Key in the following: 6.877 EEX 4 CHS "m""in" CONVERT ENTER. The stack will show 2: 0.027 1: "in".

The figure mH/1000T is used to select the specific inductance of the core. For this example, core # D-42616-16 with an A_L of 160 should be selected, which is the closest to 172. Hence an air gap of 0.027 in. corresponds to a specific inductance of 172, while the core catalog shows an A_L of 160 with an air gap of from 0.031 to 0.033 in.

Experience shows that several build iterations will be needed for the final design, thus these numbers provide a good starting point.

Changing the wire current density (cmA) to 750 c.m./A instead of 208 and using the same 2616 core gives the following output: (Same inputs as above; Ac = 0.94, MLT = 5.2):

```
6:  "Ap(m^4) = 1.759E-8"
5:  "N = 15."
4:  "AWG = 11."
3:  "Pcu(W) = 0.349"
2:  "Gap = 6.877E-4"
1:  "mH/1000T = 172."
```

The decreased current density forces a larger wire size with an attendant reduction in copper loss. A higher current density or larger core may have to be chosen, as 15 turns of 11 ga. wire may not fit in the bobbin. This can be determined from graphs given in the core catalog.

Trying a 2823 core with an Ap of 7.550E-9, Ac = 1.28, and a MLT = 5.70 gives the following output:

```
6:  "Ap(m^4) = 1.759E-8"
5:  "N = 11."
4:  "AWG = 11."
3:  "Pcu(W) = 0.281"
2:  "Gap = 5.050E-4"
1:  "mH/1000T = 319."
```

The turns count goes down to 11 with the larger core, which may fit. The mH/1000T of 319 is closest to the core with an A_L of 250, but the next higher value of 400 should be chosen to insure the required inductance can be obtained. The gap size of 5.05E-4 meters converts to 0.020 in. This is between the catalog gap size of 0.027 in. for A_L = 250 and 0.014 in. for A_L = 400.

The core loss must be determined from mW/cm³ curves given in the core catalog.

d. Listing

Program Listing	Comments
'COIL'	
<< LVL3 cmA INV CMDM2 'J' STO 'L*I^2/(Bm*J*Ku)'	Get Ap
EVAL 3 SCI "Ap(m^4) = " SWAP ->STR + 'S1' STO	
S1 HALT DROP NTWS PCU GAP 1000 DUP N / SQ L	Get N, Pcu, air gap, A_L, & purge
* * 0 FIX RND "mH/1000T = " SWAP ->STR + 'S6'	
STO S1 S2 S3 S4 S5 S6 (S1 S2 S3 S4 S5 S6 N J)	
PURGE 3 FIX UP 20 MENU >>	
'NTWS'	No. of Turns & Wire Size
<< CLLCD "Ac(cm^2) ?" 1 DISP HALT 'Ac' STO	
L I * Bm Ac "cm^2" "m^2" CONVERT DROP * /	
'N' STO N 0 FIX "N = " SWAP ->STR + 'S2'	Store display strings
STO I J / CMDM2 WGA ->STR "AWG = " SWAP +	
'S3' STO >>	
'PCU'	Get copper loss
<< CLLCD 'MLT(cm) ?" 1 DISP HALT 'MLT' STO	
CLOSS 3 FIX ->STR "Pcu(W) = " SWAP + 'S4'	
STO >>	Get Pcu string

```
'CLOSS'                                          Copper loss
<< I Rhocu * J * MLT 'cm' 'm' CONVERT DROP
* N * >>

'WGA'                                            Get wire gauge
<< sqrt pi * LOG 20 * NEG 60 + >>

'GAP'                                            Get air gap
<< uo N SQ * Ac "cm ^ 2" "m ^ 2" CONVERT
DROP * L / "Gap = " 3 SCI SWAP ->STR +
'S5' STO >>

'CMDM2'                                          c.m.
                                                 ---  conversion
<< "m^2" "mil^2" CONVERT DROP 4 * pi / >>        m²
```

e. References:

1. Advances in Switched-Mode Power Conversion, Vols I & II, R.D. Middlebrook, Slobodan Cuk, TESLAco, 1983, p. 303.

2. Ferrite Cores for Power and Filter Applications, Magnetics, Inc., Catalog No. FC405A, 1988.