HEWLETT-PACKARD

Step-by-Step Examples for Your HP-28C

Vectors and Matrices

Vectors and Matrices

Step-by-Step Examples for Your HP-28C

HEWLETT PACKARD

Edition 1 March 1987
Reorder Number 00028-90044

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsiblity for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

© 1987 by Hewlett-Packard Co.

This document contains proprietary information that is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company.

Portable Computer Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Printing History

Edition 1 March 1987 Mfg No. 00028-90059

Welcome...

... to the HP-28C Step-by-Step Booklets. These books are designed to help you get the most from your HP-28C calculator.

This booklet, *Vectors and Matrices*, provides examples and techniques for solving problems on your HP-28C. A variety of matrix manipulations are included, designed to familiarize you with the many functions built into your HP-28C.

Before you try the examples in this book, you should be familiar with certain concepts from the owner's documentation:

- The basics of your calculator – how to move from menu to menu, how to exit graphics and edit modes, and how to use the menu to assign values to, and solve for, user variables.
- Entering numbers and algebraic expressions into the calculator.

Please review the section "How To Use This Booklet." It contains important information on the examples in this booklet.

For more information about the topics in the *Vectors and Matrices* booklet, refer to a basic textbook on the subject. Many references are available in university libraries and in technical and college bookstores. The examples in the booklet demonstrate approaches to solving certain problems, but they do not cover the many ways to approach solutions to mathematical problems.

Our thanks to Brenda C. Bowman of Oregon State University for developing the problems in this book.

Contents

How To Use This Booklet

Please take a moment to familiarize yourself with the formats used in this booklet.

Keys and Menu Selection

A box represents a key on the calculator keyboard:

[ENTER]
[1/x]
[STO]

[ARRAY]
[PLOT]
[ALGEBRA]

In many cases, a box represents a shifted key on the HP-28C. In the example problems, the shift key is NOT explicitly shown (for example, [ARRAY] requires the press of the shift key, followed by the ARRAY key, found above the "A" on the left keyboard).

The "inverse" highlight represents a menu label:

DRAW (found in the [PLOT] menu)
ISOL (found in the [ALGEBRA] menu)
ABCD (a user-created name, found in the [USER] menu)

Menus typically include more menu labels than can be displayed above the six redefinable menu keys. Press [NEXT] and [PREV] to roll through the menu options. For simplicity, [NEXT] and [PREV] are NOT shown in the examples.

Solving for a user variable within [SOLVR] is initiated by the shift key, followed by the appropriate user-defined menu key:

[ ] [ABCD].

The keys above indicate the shift key, followed by the user-defined key labeled "ABCD". Pressing these keys initiates the Solver function to seek a solution for "ABCD" in a specified equation.

The symbol [< >] indicates the cursor-menu key.

Interactive Plots and the Graphics Cursor

Coordinate values you obtain from plots using the [INS] and [DEL] digitizing keys may differ from those shown, due to small differences in the positions of the graphics cursor. The values you obtain should be satisfactory for the Solver root-finding that follows.

Display Formats and Numeric Input

Negative numbers, displayed as

```
-5
-12345.678
[[-1,-2,-3 [ -4,-5,-6 [ ...
```

are created using the [CHS] key:

```
5 [CHS]
12345.678 [CHS]
[[1 [CHS] ,2 [CHS] , ...
```

The examples in this book typically specify a display format for the number of decimal places. If your display is set such that numeric displays do not match exactly, you can modify your display format with the [MODE] menu and the [FIX] key within that menu (e.g. [MODE] 2 [FIX]).

General Matrix Operations

This section illustrates several basic matrix manipulations found in common matrix problems, including addition, matrix multiplication, determinants, and so forth. Also included are several programs that demonstrate operations on matrix minors and rank.

Sum of Matrices

This example illustrates two methods for creating a matrix.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

$$B = \begin{bmatrix} 2 & -3 & 0 & 1 \\ 0 & 4 & -1 & 2 \\ 1 & -3 & 2 & -2 \end{bmatrix}$$

Compute $A + B$.

CLEAR < >

```
4:
3:
2:
1:
```

Key in the elements of matrix A in row order form. Put each element on the stack individually.

1 ENTER
2 ENTER
3 ENTER
4 ENTER
5 ENTER
6 ENTER
7 ENTER
8 ENTER
9 ENTER
10 ENTER
11 ENTER
12 ENTER

```
4:                    9
3:                   10
2:                   11
1:                   12
```

Key in the dimensions $\{m, n\}$ of matrix A. Remember to use a space to separate the two numbers.

{3 4} ENTER

```
4:                   10
3:                   11
2:                   12
1:              { 3 4 }
```

Put the stack elements into the matrix.

ARRAY
→ARRY

```
1: [[ 1 2 3 4 ]
    [ 5 6 7 8 ]
    [ 9 10 11 12 ]]
→ARRY ARRY→ PUT GET PUTI GETI
```

Store the matrix in A for the next problem section.

'A STO

```
3:
2:
1:
→ARRY ARRY→ PUT GET PUTI GETI
```

Enter matrix B, using a space to separate the matrix elements. Note the two different methods used to enter the elements of A and B.

```
[[2 -3 0 1[0 4 -1 2
[1 -3 2 -2 ENTER
```

```
1: [[ 2 -3 0 1 ]
    [ 0 4 -1 2 ]
    [ 1 -3 2 -2 ]]
→ARRY ARRY→ PUT GET PUTI GETI
```

Compute the sum $A + B$.

A ENTER

```
1: [[ 1 2 3 4 ]
    [ 5 6 7 8 ]
    [ 9 10 11 12 ]]
→ARRY ARRY→ PUT GET PUTI GETI
```

+

```
1: [[ 3 -1 3 5 ]
    [ 5 10 6 10 ]
    [ 10 7 13 10 ]]
→ARRY ARRY→ PUT GET PUTI GETI
```

Matrix Multiplication

Compute the product of two matrices, The first matrix must have dimensions $k \times m$, the second matrix has dimensions $m \times n$, and the product has dimensions $k \times n$. In this example, $k = 3$, $m = 4$, and $n = 2$.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

$$D = \begin{bmatrix} -1 & 1 \\ 2 & 4 \\ -2 & 3 \\ 5 & 4 \end{bmatrix}$$

Compute $A * D$.

Enter the 3×4 matrix A from the previous example.

A [ENTER]

```
1: [[ 1 2 3 4 ]
    [ 5 6 7 8 ]
    [ 9 10 11 12 ]]
→ARRY ARRY→ PUT GET PUTI GETI
```

Enter the 3×2 matrix D.

`[[-1 1[2 4[-2 3[5 4`
[ENTER]

```
1: [[ -1 1 ]
    [ 2 4 ]
    [ -2 3 ]
→ARRY ARRY→ PUT GET PUTI GETI
```

Compute the product $A * D$.

[×]

```
1: [[ 17 34 ]
    [ 33 82 ]
    [ 49 130 ]]
→ARRY ARRY→ PUT GET PUTI GETI
```

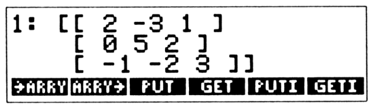
Determinant of a Matrix

Solve for the determinant of an $n \times n$ matrix.

$$A = \begin{bmatrix} 2 & -3 & 1 \\ 0 & 5 & 2 \\ -1 & -2 & 3 \end{bmatrix}$$

Key in the 3×3 matrix.

[CLEAR]
`[[2 -3 1[0 5 2[-1 -2 3`
[ENTER] [ARRAY]



Compute det(A).

DET

The determinant is 49.

Inverse of a Matrix

Compute the inverse of a square $n \times n$ matrix.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

Clear the stack and set the number display mode to two decimal places.

CLEAR
MODE 2 FIX

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the elements of the 3×3 matrix.

[[1 2 3[2 4 5[3 5 6
ENTER

```
1: [[ 1.00 2.00 3.00 ]
    [ 2.00 4.00 5.00 ]
    [ 3.00 5.00 6.00 ]]
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Compute A^{-1}.

1/x

```
1: [[ 1.00 -3.00 2.00 ]
    [ -3.00 3.00 -1.00…
    [ 2.00 -1.00 6.66E…
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Transpose of a Matrix

Compute the transpose of an $m \times n$ matrix A. A^T will be of dimension $n \times m$.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Clear the display and set the mode to standard. Key in the 3×2 matrix A.

CLEAR
MODE
STD
`[[1 2[3 4[5 6` ENTER

```
1:  [[ 1 2 ]
     [ 3 4 ]
     [ 5 6 ]]
[ STD ] FIX  SCI  ENG  DEG [ RAD ]
```

Compute A^T.

ARRAY
TRN

```
2:
1:  [[ 1 3 5 ]
     [ 2 4 6 ]]
 SIZE  RDM  TRN  CON  IDN  RSD
```

A^T is a 2×3 matrix.

Conjugate of a Complex Matrix

Compute the conjugate conj(A) of the complex matrix A.

$$A = \begin{bmatrix} 1+3i & i \\ 3 & 2-4i \end{bmatrix}$$

[CLEAR]

```
3:
2:
1:
SIZE RDM TRN CON IDN RSD
```

Key in the elements individually in row order form. Each pair represents (real part, imaginary part). Note the commas in the keystrokes below may be used alternately with spaces.

`(1,3` [ENTER]

```
3:
2:
1:                 (1,3)
SIZE RDM TRN CON IDN RSD
```

`(0,1` [ENTER]

```
3:
2:                 (1,3)
1:                 (0,1)
SIZE RDM TRN CON IDN RSD
```

`(3,0` [ENTER]

```
3:                 (1,3)
2:                 (0,1)
1:                 (3,0)
SIZE RDM TRN CON IDN RSD
```

`(2,-4` [ENTER]

```
3:                 (0,1)
2:                 (3,0)
1:                (2,-4)
SIZE RDM TRN CON IDN RSD
```

Key in the dimensions of the matrix.

`{2 2}` [ENTER]

```
3:                 (3,0)
2:                (2,-4)
1:               { 2 2 }
SIZE RDM TRN CON IDN RSD
```

Place the stack elements in an array.

[ARRAY]
→ARRY

```
2:
1: [[ (1,3) (0,1) ]
    [ (3,0) (2,-4) ]]
→ARRY ARRY→ PUT GET PUTI GETI
```

Compute the conjugate.

CONJ

```
2:
1: [[ (1,-3) (0,-1) ]
    [ (3,0) (2,4) ]]
R→C C→R RE IM CONJ NEG
```

Minor of a Matrix

The minor M_{ij} is formed by removing row i and column j from matrix A, then computing det (M_{ij}). A program is written to perform this function for any $n \times n$ matrix.

Program ROW below is a subroutine used to remove a row or column from a matrix.

Program Listing	Explanation
SWAP ARRY→ LIST→	Swap the matrix into level 1, then separate the matrix into individual elements and its dimension.
DROP	Drop the number of items in the list.
→ n m «	Save the row and column in n and m.
n DUP m × 2 +	Compute offset to row (col) number on stack.
ROLL − m × →LIST → list	Place $(n - i)*m$ elements into list.
« m DROPN	Drop row i (col j) from stack.
list » LIST→	Separate temporary list into individual elements.
DROP	Drop number of list elements.
n 1 − m 2 →LIST →ARRY	Reconstruct matrix with row (col) removed.

Program MINOR utilizes the subroutine ROW to remove a row, and then a column, from the matrix.

Program Listing	Explanation
3 ROLLD	Roll down the matrix and row i.
ROW TRN	Remove row i and transpose for column removal.
SWAP ROW TRN	Remove column j and transpose back.

Key in the program ROW.

CLEAR

```
« SWAP ARRY→ LIST→
DROP → n m « n DUP
m × 2 + ROLL - m × →LIST
→ list « m DROPN list »
LIST→ DROP n 1 - m 2
→LIST →ARRY ENTER <>
```

```
1: « SWAP ARRY→ LIST→
   DROP → n m « n DUP m
   * 2 + ROLL - m *
   →LIST → list « m
```

Store the program ROW.

'ROW STO

```
4:
3:
2:
1:
```

Key in the program MINOR.

```
« 3 ROLLD ROW TRN SWAP
ROW TRN ENTER <>
```

```
3:
2:
1: « 3 ROLLD ROW TRN
   SWAP ROW TRN »
```

Store the program MINOR.

'MINOR STO

```
4:
3:
2:
1:
```

Compute M_{23} of the following matrix.

$$A = \begin{bmatrix} 2 & -3 & 4 & -4 \\ 6 & 5 & 2 & -1 \\ 1 & 0 & 3 & -2 \\ 0 & -5 & 3 & -6 \end{bmatrix}$$

Enter the matrix.

```
[[2 -3 4 -4[6 5 2 -1[1 0
 3 -2[0 -5 3 -6 ENTER
```

```
1: [[ 2 -3 4 -4 ]
    [ 6 5 2 -1 ]
    [ 1 0 3 -2 ]
    [ 0 -5 3 -6 ]]
```

Enter the row and column to be removed.

2 [ENTER] 3 [ENTER]

```
4:
3: [[ 2 -3 4 -4 ]  [ 6...
2:                      2
1:                      3
```

Compute M_{23}.

[USER] [MINO]

```
1: [[ 2 -3 -4 ]
    [ 1  0 -2 ]
    [ 0 -5 -6 ]]
MINO ROW
```

Compute the minor det (M_{ij}).

[ARRAY] [DET]

```
3:
2:
1:                    -18
CROSS DOT DET ABS RNRM CNRM
```

The minor det(M_{23}) is -18.

Compute Rank

The dimension of the largest square submatrix whose determinant is non-zero is called the rank of the matrix. The rank is the maximum number of linearly independent row and column vectors.

Find the rank of matrix A.

$$A = \begin{bmatrix} 4 & 2 & -1 \\ 0 & 5 & -1 \\ 12 & -4 & -1 \end{bmatrix}$$

Program MDET below is used to obtain the determinant of an arbitrary matrix minor. This program uses the program MINOR from the previous problem section.

Program Listing	**Explanation**
3 PICK	Duplicate the matrix.
3 ROLLD MINOR	Produce the matrix minor.
DET	Compute the minor determinant.

Key in the program.

[CLEAR]

```
3:
2:
1:  « 3 PICK 3 ROLLD
    MINOR DET »
```

« 3 PICK 3 ROLLD MINOR DET [ENTER] [<>]

Store the program in MDET.

'MDET [STO]

```
4:
3:
2:
1:
```

Key in the matrix.

[[4 2 -1[0 5 -1 [12 -4 -1 [ENTER]

```
2:
1:  [[ 4 2 -1 ]
     [ 0 5 -1 ]
     [ 12 -4 -1 ]]
```

Make a copy of the matrix and compute the determinant to determine whether the rank = n = 3.

ENTER ARRAY DET

```
3:
2: [[ 4 2 -1 ]  [ 0 5 …
1:          .000000000048
CROSS DOT DET ABS RNRM CNRM
```

Det(A) is zero (approximately), so rank(A) is not equal to 3.

Discard det(A).

DROP <>

```
2:
1: [[ 4 2 -1 ]
    [ 0 5 -1 ]
    [ 12 -4 -1 ]]
```

Compute the minor for the 2×2 submatrices of A, until a minor is found that is not equal to zero.

Compute det M_{11}.

1 ENTER ENTER
USER MDET

```
3:
2: [[ 4 2 -1 ]  [ 0 5 …
1:                    -9
MDET MINO ROW
```

Det(M_{11}) is equal to -9, so rank(A) is equal to 2.

You may elect to purge programs ROW, MDET and MINO before continuing to the next problem sections.

Hermitian Matrices

Determine whether a matrix is Hermitian. A square matrix with real or complex elements is Hermitian if the matrix is equal to its conjugate transpose.

Determine whether the 4×4 matrix A is Hermitian.

$$A = \begin{bmatrix} 1 & 2-i & 2 & -3+i \\ 2+i & 3 & i & 3 \\ 2 & -i & 4 & 1-i \\ 3-i & 3 & 1+i & 0 \end{bmatrix}$$

Put the elements of A on the stack individually.

Keys	Display
[CLEAR] [< >] `1` [ENTER] `(2,-1` [ENTER] `2` [ENTER] `(-3,1` [ENTER]	4: 1 3: (2,-1) 2: 2 1: (-3,1)
`(2,1` [ENTER] `3` [ENTER] `(0,1` [ENTER] `3` [ENTER]	4: (2,1) 3: 3 2: (0,1) 1: 3
`2` [ENTER] `(0,-1` [ENTER] `4` [ENTER] `(1,-1` [ENTER]	4: 2 3: (0,-1) 2: 4 1: (1,-1)
`(3,-1` [ENTER] `3` [ENTER] `(1,1` [ENTER] `0` [ENTER]	4: (3,-1) 3: 3 2: (1,1) 1: 0

Enter the dimensions of A.

{4 4 ENTER

```
4:                    3
3:                (1,1)
2:                    0
1:              { 4 4 }
```

Place the elements into the matrix.

ARRY
→ARRY

```
1: [[ (1,0) (2,-1) (2,…
    [ (2,1) (3,0) (0,1…
    [ (2,0) (0,-1) (4,…
→ARRY ARRY→ PUT GET PUTI GETI
```

You can view the entire matrix to check for correctness using EDIT or VIEW↓.

Make a copy of the matrix.

ENTER

```
1: [[ (1,0) (2,-1) (2,…
    [ (2,1) (3,0) (0,1…
    [ (2,0) (0,-1) (4,…
→ARRY ARRY→ PUT GET PUTI GETI
```

Compute the conjugate transpose. Since A is complex, function TRN performs both the transpose and the conjugation.

TRN

```
1: [[ (1,0) (2,-1) (2,…
    [ (2,1) (3,0) (0,1…
    [ (2,0) (0,-1) (4,…
SIZE RDM TRN CON IDN RSD
```

Test conj(A^T) and A for equivalency. If A is Hermitian, conj(A^T) and A will be equal, and SAME will return a true flag(1).

TEST SAME

```
3:
2:
1:                    0
AND OR XOR NOT SAME ==
```

Matrix A is not Hermitian.

Systems of Linear Equations

One of the most frequent and fundamental applications of matrices arises from the need to solve a system of m linear equations in n unknowns. The HP-28C can be used to find solutions to both non-homogeneous and homogeneous systems of the form $AX = B$.

Non-Homogeneous System

Solve a system of linear equations of the form $AX = B$.

$$\begin{aligned} x_1 + x_2 - 2x_3 + x_4 + 3x_5 &= 1 \\ 3x_1 + 2x_2 - 4x_3 - 3x_4 - 8x_5 &= 2 \\ 2x_1 - x_2 + 2x_3 + 2x_4 + 5x_5 &= 3 \end{aligned}$$

Clear the stack and set the display mode to two decimal places.

CLEAR
MODE 2 FIX

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the coefficients of the system of equations.

`[[1 1 -2 1 3[3 2 -4 -3 -8 [2 -1 2 2 5` ENTER

```
1: [[ 1.00 1.00 -2.00 …
    [ 3.00 2.00 -4.00 …
    [ 2.00 -1.00 2.00 …
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Store matrix A.

'A STO

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the elements of B.

`[[1[2[3` ENTER

```
1: [[ 1.00 ]
    [ 2.00 ]
    [ 3.00 ]]
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Store matrix B.

'B STO

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

To solve for X, we use the method

$$X = \frac{A^T B}{A^T A}$$

Compute A^T.

ARRAY
A ENTER

```
1: [[ 1.00 1.00 -2.00 …
    [ 3.00 2.00 -4.00 …
    [ 2.00 -1.00 2.00 …
→ARRY ARRY→ PUT GET PUTI GETI
```

TRN

```
1: [[ 1.00 3.00 2.00 ]
    [ 1.00 2.00 -1.00 ]
    [ -2.00 -4.00 2.00…
SIZE RDM TRN CON IDN RSD
```

Multiply by B.

B ×

```
1: [[ 13.00 ]
    [ 2.00 ]
    [ -4.00 ]
SIZE RDM TRN CON IDN RSD
```

Compute A^T.

A ENTER

```
1: [[ 1.00 3.00 2.00 ]
    [ 1.00 2.00 -1.00 ]
    [ -2.00 -4.00 2.00…
SIZE RDM TRN CON IDN RSD
```

TRN

```
1: [[ 1.00 3.00 2.00 ]
    [ 1.00 2.00 -1.00 ]
    [ -2.00 -4.00 2.00…
SIZE RDM TRN CON IDN RSD
```

Multiply by A.

A ×

```
1: [[ 14.00 5.00 -10.0…
    [ 5.00 6.00 -12.00…
    [ -10.00 -12.00 24…
SIZE RDM TRN CON IDN RSD
```

Divide A^TB by A^TA.

÷

```
1: [[ 1.12 ]
    [ 1.24 ]
    [ 0.80 ]
SIZE RDM TRN CON IDN RSD
```

VIEW↑ and VIEW↓ can be used to display all of the elements. They are $x_1 = 1.12$, $x_2 = 1.24$, $x_3 = 0.80$, $x_4 = -0.08$, and $x_5 = 0.11$.

Homogeneous System

Solve a homogeneous system of linear equations of the form $AX = 0$.

$$\begin{aligned} x_1 - 2x_2 + 3x_3 &= 0 \\ 2x_1 + 6x_2 + x_3 &= 0 \\ 3x_1 - 4x_2 + 8x_3 &= 0 \end{aligned}$$

The program UT below takes a stack of vectors representing homogeneous simultaneous equations and transforms them to upper triangular form.

Program Listing	Explanation
DUP SIZE LIST→	Save number of elements
DROP → s	as s.
« s 2	For j = s (down) to 2, transform the bottom j − 1 vectors.
FOR j s j −	m = s − j + 1
1 + → m	
« 1 j 1 −	Loop for i = 1 to j − 1
FOR i i ROLL j PICK	
m 1 →LIST DUP2 GET 4 PICK	Transform the vectors.
ROT GET SWAP ÷ × −	
i ROLLD NEXT » −1 STEP »	

Key in the program.

CLEAR

```
«  DUP SIZE LIST→
DROP → s «  s 2
FOR j s j - 1 +
→ m «  1 j 1 -
FOR i i ROLL j PICK
m 1 →LIST DUP2 GET
4 PICK ROT GET SWAP
÷ × - i ROLLD
NEXT » -1 STEP »
```

ENTER <>

```
1: « DUP SIZE LIST→
   DROP → s « s 2 FOR j
   s j - 1 + → m « 1 j
   1 - FOR i i ROLL j
```

Store the program in UT.

'UT [STO]

```
4:
3:
2:
1:
```

Set the display mode to one decimal place.

[MODE] 1 [FIX]

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the coefficients.

[[1 -2 3[2 6 1[3 -4 8
[ENTER]

```
1: [[ 1.0 -2.0 3.0 ]
    [ 2.0 6.0 1.0 ]
    [ 3.0 -4.0 8.0 ]]
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Store the matrix in *ARR* for a verification at the end of the problem.

'ARR [STO]

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Edit matrix *ARR* to reduce to row echelon form.

[USER]
[ARR]

```
1: [[ 1.0 -2.0 3.0 ]
    [ 2.0 6.0 1.0 ]
    [ 3.0 -4.0 8.0 ]]
 ARR  UT
```

Use [EDIT] mode and the [DEL] key to remove the outer brackets of the array *ARR* and place the rows into three independent row vectors. After removing the left- and right-most braces, the edited rows are [ENTER]ed:

[1 -2 3]
[2 6 1]
[3 -4 8] [ENTER]

```
3:      [ 1.0 -2.0 3.0 ]
2:       [ 2.0 6.0 1.0 ]
1:      [ 3.0 -4.0 8.0 ]
 ARR  UT
```

Now transform the matrix to upper triangular form.

[UT]

```
3:      [ 1.0 -2.0 3.0 ]
2:    [ 0.0 10.0 -5.0 ]
1:       [ 0.0 0.0 0.0 ]
 ARR  UT
```

The matrix is now in row echelon form, so the system of three transformed equations is ready to be solved. The matrix represents the system of linear equations

$$\begin{aligned} x_1 - 2x_2 + 3x_3 &= 0 \\ 10x_2 - 5x_3 &= 0 \\ 0 &= 0 \end{aligned}$$

Drop the equation $0=0$.

DROP

```
3:
2:      [ 1.0 -2.0 3.0 ]
1:     [ 0.0 10.0 -5.0 ]
 ARR | UT |  |  |  |
```

Enter the equation from row 2.

'10×X2 - 5×X3=0 ENTER

```
3:      [ 1.0 -2.0 3.0 ]
2:     [ 0.0 10.0 -5.0 ]
1:        '10*X2-5*X3=0'
 ARR | UT |  |  |  |
```

Solve the equation in terms of x_3.

'X3 ENTER

```
3:     [ 0.0 10.0 -5.0 ]
2:        '10*X2-5*X3=0'
1:                  'X3'
 ARR | UT |  |  |  |
```

Isolate the term x_3.

ALGEBRA
ISOL

```
3:      [ 1.0 -2.0 3.0 ]
2:     [ 0.0 10.0 -5.0 ]
1:             '10*X2/5'
TAYLR|ISOL|QUAD|SHOW|OBGET|EXGET
```

Collect terms.

COLCT

```
3:      [ 1.0 -2.0 3.0 ]
2:     [ 0.0 10.0 -5.0 ]
1:                '2*X2'
COLCT|EXPAN|SIZE|FORM|OBSUB|EXSUB
```

The solution is $x_3=2{*}x_2$. Remove row 2 to solve row 1.

DROP
DROP

```
3:
2:
1:      [ 1.0 -2.0 3.0 ]
COLCT|EXPAN|SIZE|FORM|OBSUB|EXSUB
```

Enter the equation for row 1, making the substitution for x_3.

'X1 − 2 × X2 + 6 × X2
[ENTER]

```
3:
2:      [ 1.0 -2.0 3.0 ]
1:       'X1-2*X2+6*X2'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Solve for x_1.

'X1 [ENTER]

```
3:      [ 1.0 -2.0 3.0 ]
2:       'X1-2*X2+6*X2'
1:                 'X1'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Isolate the term.

[ISOL]

```
3:
2:      [ 1.0 -2.0 3.0 ]
1:       '-(6*X2)+2*X2'
TAYLR ISOL QUAD SHOW OBGET EXGET
```

Collect terms.

[COLCT]

```
3:
2:      [ 1.0 -2.0 3.0 ]
1:             '-(4*X2)'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

The result is $x_1 = -4*x_2$. A solution is $x_1 = -4, x_2 = 1, x_3 = 2$. Verify this 3×1 solution vector X. Key in vector X.

[[−4[1[2 [ENTER]

```
1: [[ -4.0 ]
    [ 1.0 ]
    [ 2.0 ]]
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Put the coefficient matrix ARR on the stack.

[USER]
[ARR]

```
1: [[ 1.0 -2.0 3.0 ]
    [ 2.0 6.0 1.0 ]
    [ 3.0 -4.0 8.0 ]]
ARR UT
```

Swap the positions of ARR and X.

[SWAP]

```
1: [[ -4.0 ]
    [ 1.0 ]
    [ 2.0 ]]
ARR UT
```

Multiply $ARR * X$.

$\boxed{\times}$

```
1: [[ 0.0 ]
    [ 0.0 ]
    [ 0.0 ]]
 ARR | UT |  |  |  |
```

$ARR * X = 0$. Thus X is a verified solution to the system.

Program UT will be used in a later problem section.

Iterative Refinement

Due to rounding errors, in some cases the numerically calculated solution Z is not precisely the solution to the original system $AX = B$. In many applications, Z may be an adequate solution. When additional accuracy is desired, the computed solution Z can be improved by the method of iterative refinement. This method uses the residual error associated with a solution to modify the solution.

Solve the system of linear equations $AX = B$.

$$A = \begin{bmatrix} 33 & 16 & 72 \\ -24 & -10 & -57 \\ -8 & -4 & -17 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Clear the display and the set the standard display mode.

CLEAR
MODE ≣STD≣

```
3:
2:
1:
[STD ] FIX  SCI  ENG  DEG [RAD ]
```

Solve for $AX = B$ and improve the accuracy by iterative refinement using residual corrections. Key in the coefficient matrix.

`[[33 16 72[-24 -10 -57`
`[-8 -4 -17` ENTER

```
1: [[ 33 16 72 ]
    [ -24 -10 -57 ]
    [ -8 -4 -17 ]]
[STD ] FIX  SCI  ENG  DEG [RAD ]
```

Store matrix A.

`'A` STO

```
3:
2:
1:
[STD ] FIX  SCI  ENG  DEG [RAD ]
```

Key in the constant matrix.

`[[0[0[1` ENTER

```
1: [[ 0 ]
    [ 0 ]
    [ 1 ]]
[STD ] FIX  SCI  ENG  DEG [RAD ]
```

Store matrix B.

'B [STO]

```
3:
2:
1:
[STD ] FIX  SCI  ENG  DEG [RAD ]
```

Compute $Z = B/A$.

[USER]
B

```
1: [[ 0 ]
    [ 0 ]
    [ 1 ]]
 B    A    UT
```

A

```
1: [[ 33 16 72 ]
    [ -24 -10 -57 ]
    [ -8 -4 -17 ]]
 B    A    UT
```

[÷]

```
1: [[ -31.9999999989 ]
    [ 25.4999999991 ]
    [ 8.99999999969 ]]
 B    A    UT
```

Store the approximate 3×1 solution matrix Z.

'Z [STO]

```
3:
2:
1:
 Z    B    A    UT
```

Compute the Residual Error Matrix R, where $R = B - AZ$. The function RSD calculates R using extended precision.

B
A
Z

```
1: [[ -31.9999999989 ]
    [ 25.4999999991 ]
    [ 8.99999999969 ]]
 Z    B    A    UT
```

Solve using the RSD function.

[ARRAY]
RSD

```
1: [[ .00000000042 ]
    [ -.00000000027 ]
    [ -.00000000007 ]]
SIZE RDM  TRN  CON  IDN  RSD
```

Store matrix R.

'R [STO]

```
3:
2:
1:
SIZE RDM  TRN  CON  IDN  RSD
```

Find the actual error $E = |Z - X| = (B - AZ)/A = R/A$.

USER
R
A
÷

```
1: [[ -1.09999999997E-…
   [ 8.99999999977E-1…
   [ 3.09999999992E-1…
 R  Z  B  A  UT
```

Compute the corrected solution $X = Z + E$.

Z
+

```
1: [[ -32 ]
   [ 25.5 ]
   [ 9 ]]
 R  Z  B  A  UT
```

X = the corrected solution.

Vector Spaces

Vector spaces are widely used in mathematics, physics and engineering to represent physical properties such as magnitude and direction within a geometric system. Several important vector operations can be performed easily using the built-in functions of the [ARRAY] menu.

Basis

A basis is a set of n linearly independent vectors that span the vector space $V_n(\mathrm{R})$.

Determine whether the vectors X_1, X_2, and X_3 form a basis that spans $V_3(\mathrm{R})$.

$$X_1 = [\ 1 \quad 1\ \ 2\ ]$$
$$X_2 = [\ 3 \quad 2\ \ 4\ ]$$
$$X_3 = [\ 1 \ \ -3\ \ 1\ ]$$

Clear the stack and set the standard display mode.

CLEAR
MODE STD

```
3:
2:
1:
[STD] FIX SCI ENG DEG [RAD]
```

Key in the three vectors as a 3 × 3 matrix A and make two copies.

`[[1 1 2[3 2 4[1 -3 1`
ENTER ENTER

```
1: [[ 1  1 2 ]
    [ 3  2 4 ]
    [ 1 -3 1 ]]
[STD] FIX SCI ENG DEG [RAD]
```

Store matrix A for the next problem section.

`'A` STO

```
3:
2:
1:
[STD] FIX SCI ENG DEG [RAD]
```

Compute det(A).

ARRAY
DET

```
3:
2:
1:          -7.00000000004
CROSS DOT DET ABS RNRM CNRM
```

Det(A) = −7. Thus A is non-singular, and the three row vectors are linearly independent and form a basis.

Orthogonality

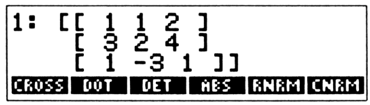
Two vectors are mutually orthogonal if their inner product equals zero.

Determine which of the vectors from the previous problem are mutually orthogonal.

CLEAR

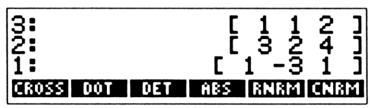
Recall matrix A to the stack.

A ENTER



Use EDIT to remove the outer brackets of the array A and form three row vectors. After removing the left- and right-most braces with DEL, the edited rows are ENTERed:

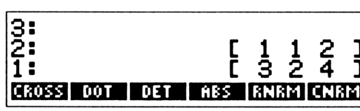
```
[ 1 1 2 ]
[ 3 2 4 ]
[ 1 -3 1 ] ENTER
```



Note: two utility routines for modifying a two-dimensional array to its row components and vice versa are shown at the end of this section. These routines can be used as alternatives for the editing shown above.

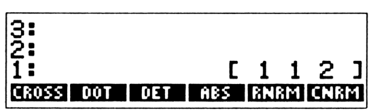
The third vector is X_3.

'X3 STO



The second vector is X_2.

'X2 STO



The first vector is X_1.

'X1 STO

```
3:
2:
1:
CROSS DOT DET ABS RNRM CNRM
```

Compute the inner products.

X1 ENTER
X2 ENTER

```
3:
2:                 [ 1 1 2 ]
1:                 [ 3 2 4 ]
CROSS DOT DET ABS RNRM CNRM
```

DOT

```
3:
2:
1:                        13
CROSS DOT DET ABS RNRM CNRM
```

$X_1 \cdot X_2 = 13$. These rows are not orthogonal.

DROP
X2 ENTER
X3 ENTER

```
3:
2:                 [ 3 2 4 ]
1:                [ 1 -3 1 ]
CROSS DOT DET ABS RNRM CNRM
```

DOT

```
3:
2:
1:                         1
CROSS DOT DET ABS RNRM CNRM
```

$X_2 \cdot X_3 = 1$. These rows are not orthogonal.

DROP
X1 ENTER
X3 ENTER

```
3:
2:                 [ 1 1 2 ]
1:                [ 1 -3 1 ]
CROSS DOT DET ABS RNRM CNRM
```

DOT

```
3:
2:
1:                         0
CROSS DOT DET ABS RNRM CNRM
```

$X_1 \cdot X_3 = 0$. These two vectors are mutually orthogonal.

Matrix Utility Programs

Several problem sections up to this point have included use of [EDIT] mode to reduce a matrix to its row elements. The following utility programs can be used as alternatives for changing a matrix to its row elements, and vice versa.

Program ROW→ below takes a stack of n row vectors and the number n in level 1 and returns the matrix combining those n row vectors.

```
«  OVER SIZE LIST→ DROP
→  n m «  0 n 1 -
FOR i i m × n i - +
ROLL ARRY→ DROP NEXT
n m 2 →LIST →ARRY » »
[ENTER] [<>]
```

```
1: « OVER SIZE LIST→
   DROP → n m « 0 n 1 -
   FOR i i m * n i - +
   ROLL ARRY→ DROP NEXT
```

After keying in the program above, store the program and put the rows of array A in matrix form.

```
'ROW→ [STO]

[1,1,2]
[3,2,4]
[1,-3,1] [ENTER]
3 [USER] ≡ROW→≡
```

```
1: [[ 1 1 2 ]
    [ 3 2 4 ]
    [ 1 -3 1 ]]
ROW→ | UT |  |  |  |
```

Program →ROW below takes a matrix and separates it into individual rows on the stack.

```
«  ARRY→ LIST→ DROP
→  n m «  1 n FOR i m 1
→LIST →ARRY n i -
m × i + ROLLD NEXT » »
[ENTER] [<>]
```

```
1: « ARRY→ LIST→ DROP →
   n m « 1 n FOR i m 1
   →LIST →ARRY n i - m
   * i + ROLLD NEXT » »
```

After keying in the program above, store the program and convert the matrix from above back to row form.

```
'→ROW [STO]
[USER] ≡→ROW≡
```

```
3:             [ 1 1 2 ]
2:             [ 3 2 4 ]
1:            [ 1 -3 1 ]
→ROW | ROW→ | UT |  |  |  |
```

Vector Length

Find the length of vector X_1 (from the previous problem section), denoted by

$$||X_1|| = \sqrt{X_1 \cdot X_1}$$

Clear the stack and set the display mode to two decimal places.

CLEAR
MODE 2 FIX

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Recall X_1 from the previous problem. Since X_1 was stored, you may alternatively use USER X1.

X1 ENTER

```
3:
2:
1:     [ 1.00 1.00 2.00 ]
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Function ABS returns the Frobenius norm of an array, which is equivalent to the length of a vector.

ARRAY
ABS

```
3:
2:
1:                  2.45
CROSS DOT  DET  ABS  RNRM CNRM
```

$|| X_1 || = 2.45.$

Normalization

To normalize a vector X into its unique unit vector U, divide each component of X by $||X||$. We will normalize X_1. Vectors X_1, X_2, and X_3 are from the section entitled "Orthogonality."

Enter a program that computes $X/||X||$.

CLEAR
« DUP ABS $\frac{1}{x}$ × » ENTER

```
3:
2:
1:      « DUP ABS INV * »
CROSS DOT DET ABS RNRM CNRM
```

Store program NORM.

'NORM STO

```
3:
2:
1:
CROSS DOT DET ABS RNRM CNRM
```

Enter the vector to be normalized.

USER
X1

```
3:
2:
1:    [ 1.00 1.00 2.00 ]
NORM X1 X2 X3 →ROW ROW→
```

Normalize the vector.

NORM

```
3:
2:
1:    [ 0.41 0.41 0.82 ]
NORM X1 X2 X3 →ROW ROW→
```

The result is U_1= [0.41 0.41 0.82].

Normalize vector X_2.

X2

```
3:
2:    [ 0.41 0.41 0.82 ]
1:    [ 3.00 2.00 4.00 ]
NORM X1 X2 X3 →ROW ROW→
```

NORM

```
3:
2:    [ 0.41 0.41 0.82 ]
1:    [ 0.56 0.37 0.74 ]
NORM X1 X2 X3 →ROW ROW→
```

The result is U_2 = [0.56 0.37 0.74].

Finally, normalize vector X_3.

X3

```
3:     [ 0.41 0.41 0.82 ]
2:     [ 0.56 0.37 0.74 ]
1:   [ 1.00 -3.00 1.00 ]
NORM  X1  X2  X3  →ROW  ROW→
```

NORM

```
3:     [ 0.41 0.41 0.82 ]
2:     [ 0.56 0.37 0.74 ]
1:   [ 0.30 -0.90 0.30 ]
NORM  X1  X2  X3  →ROW  ROW→
```

The result is $U_3 = [0.30\ -0.90\ 0.30]$.

Gram-Schmidt Orthogonalization

Form an orthogonal basis that spans $V_3(\mathrm{R})$ using the Gram-Schmidt process. Given that X_1, X_2, and X_3 form a basis, then the vectors Y_1, Y_2, and Y_3 form an orthogonal basis, formed by the following process.

$$Y_1 = X_1$$

$$Y_2 = X_2 - \left(\frac{Y_1 \cdot X_2}{Y_1 \cdot Y_1} * Y_1 \right)$$

$$Y_3 = X_3 - \left(\frac{Y_2 \cdot X_3}{Y_2 \cdot Y_2} * Y_2 \right) - \left(\frac{Y_1 \cdot X_3}{Y_1 \cdot Y_1} * Y_1 \right)$$

Vectors X_1, X_2, and X_3 are from the section entitled "Orthogonality".

Calculate Y_1.

CLEAR
USER X1

```
3:
2:
1:     [ 1.00 1.00 2.00 ]
 X1  X2  X3  UT
```

Store Y_1.

'Y1 STO

```
3:
2:
1:
 Y1  X1  X2  X3
```

Write a program to calculate Y_2.

« X2 Y1 X2 DOT Y1 Y1
DOT ÷ Y1 × - »
ENTER

```
2:
1: « X2 Y1 X2 DOT Y1 Y1
   DOT / Y1 * - »
 Y1  X1  X2  X3
```

Execute the program.

EVAL

```
3:
2:
1: [ 0.83 -0.17 -0.33 ]
 Y1  X1  X2  X3
```

$Y_2 = [0.83\ -0.17\ -0.33]$.   Store Y_2.

'Y2 [STO]

```
3:
2:
1:
 Y2 | Y1 | X1 | X2 | X3 |
```

Write a program to calculate Y_3.

« X3 Y2 X3 DOT Y2 Y2 DOT ÷ Y2 × - Y1 X3 DOT Y1 Y1 DOT ÷ Y1 × - » [ENTER]

```
1: « X3 Y2 X3 DOT Y2 Y2
   DOT / Y2 * - Y1 X3
   DOT Y1 Y1 DOT / Y1 *
 Y2 | Y1 | X1 | X2 | X3 |
```

Execute the program.

[EVAL]

```
3:
2:
1: [ 4.00E-12 -2.80 1....
 Y2 | Y1 | X1 | X2 | X3 |
```

$Y_3 = [4.0E-12\ -2.80\ 1.40]$.   Store Y_3.

'Y3 [STO]

```
3:
2:
1:
 Y3 | Y2 | Y1 | X1 | X2 | X3
```

The vectors Y_1, Y_2, and Y_3 form an orthogonal basis.

Generalized Gram-Schmidt Orthogonalization Routine

The program GSO below is a generalized routine for finding an orthogonal basis for an arbitrary list of vectors.

```
«  DUP SIZE LIST→ DROP
DUP DUP 2 + ROLLD →LIST
→ M «  2 SWAP FOR n M
n GET 1 n 1 - FOR i M i
GET DUP DUP2 DOT INV ×
SWAP 3 PICK DOT × -
NEXT n M SWAP ROT PUT 'M'
STO NEXT M LIST→
DROP » » [ENTER] [<>]
```

```
1: « DUP SIZE LIST→
   DROP DUP DUP 2.00 +
   ROLLD →LIST → M «
   2.00 SWAP FOR n M n
```

After keying in the program above, store the program and form an orthogonal basis for the three vectors in the previous example.

```
'GSO [STO]

[1,1,2]
[3,2,4]
[1,-3,1] [USER] [GSO]
```

```
3:    [ 1.00 1.00 2.00 ]
2: [ 0.83 -0.17 -0.33 ]
1: [ 4.00E-12 -2.80 1.…
 GSO  Y3  Y2  Y1  X1  X2
```

Orthonormal Basis

Form an orthonormal basis G_i of orthogonal unit vectors that spans $V_3(R)$. Vectors Y_1, Y_2, and Y_3 and program NORM are from the two previous problem sections.

$$G_i = \frac{Y_i}{||Y_i||}$$

Calculate G_1.

[CLEAR]
[USER] ≡Y1≡

```
3:
2:
1:     [ 1.00 1.00 2.00 ]
NORM  Y3  Y2  Y1  X1  X2
```

Execute the normalization program from the section entitled "Normalization".

≡NORM≡

```
3:
2:
1:     [ 0.41 0.41 0.82 ]
NORM  Y3  Y2  Y1  X1  X2
```

Store the result in G_1.

'G1 [STO]

```
3:
2:
1:
G1  NORM  Y3  Y2  Y1  X1
```

Calculate G_2.

≡Y2≡

```
3:
2:
1: [ 0.83 -0.17 -0.33 ]
G1  NORM  Y3  Y2  Y1  X1
```

Compute the norm.

≡NORM≡

```
3:
2:
1: [ 0.91 -0.18 -0.37 ]
G1  NORM  Y3  Y2  Y1  X1
```

Store the result in G_2.

'G2 [STO]

```
3:
2:
1:
G2  G1  NORM  Y3  Y2  Y1
```

Calculate G_3.

Y3

```
3:
2:
1: [ 4.00E-12 -2.80 1.…
 G2 | G1 |NORM| Y3 | Y2 | Y1
```

Compute the norm.

NORM

```
3:
2:
1: [ 1.28E-12 -0.89 0.…
 G2 | G1 |NORM| Y3 | Y2 | Y1
```

Store the result in G_3.

'G3 [STO]

```
3:
2:
1:
 G3 | G2 | G1 |NORM| Y3 | Y2
```

Verify that all three vectors are mutually orthogonal.

G1
G2

```
3:
2:    [ 0.41 0.41 0.82 ]
1: [ 0.91 -0.18 -0.37 ]
 G3 | G2 | G1 |NORM| Y3 | Y2
```

Compute the dot product ($G_1 \cdot G_2$).

[ARRAY]
DOT

```
3:
2:
1:               -8.98E-12
CROSS| DOT | DET | ABS |RNRM|CNRM
```

$G_1 \cdot G_2 \approx 0$.

Compute the dot product ($G_2 \cdot G_3$).

[DROP]
[USER]
G2
G3
[ARRAY]
DOT

```
3:
2:
1:                5.18E-13
CROSS| DOT | DET | ABS |RNRM|CNRM
```

$G_2 \cdot G_3 \approx 0$.

Compute the dot product ($G_1 \cdot G_3$).

DROP
USER
G1
G3
ARRAY
DOT

```
3:
2:
1:                  2.97E-12
CROSS DOT  DET  ABS  RNRM CNRM
```

$G_1 \cdot G_3 \approx 0$.

Thus, since all three dot products are approximately equal to zero, the three vectors are mutually orthogonal.

Now verify that they form a basis. Combine the three vectors into one array by placing the elements on the stack and removing their individual dimension lists.

DROP
USER G1
ARRAY ARRY→
DROP
USER G2
ARRAY ARRY→
DROP
USER G3
ARRAY ARRY→
DROP

```
3:                  1.28E-12
2:                     -0.89
1:                      0.45
→ARRY ARRY→ PUT  GET  PUTI GETI
```

Note the utility program →ROW, described in the section entitled "Orthogonality," could also be used to form the list of vectors above.

Next key in the dimensions of the matrix that will be formed by the three vectors.

{ 3 3 } ENTER

```
3:                     -0.89
2:                      0.45
1:           { 3.00 3.00 }
→ARRY ARRY→ PUT  GET  PUTI GETI
```

Finally, place the three vectors into matrix form.

→ARRY

```
1: [[ 0.41 0.41 0.82 ]
    [ 0.91 -0.18 -0.37…
    [ 1.28E-12 -0.89 0…
→ARRY ARRY→ PUT  GET  PUTI GETI
```

Compute the determinant.

DET

The determinant is -1. The matrix is non-singular and the vectors form an orthonormal basis.

Purge the vectors $X_1, X_2, X_3, Y_1, Y_2, Y_3, G_1, G_2, G_3$ and program NORM.

Eigenvalues

Another fundamental use for matrices is in developing a structure to represent linear transformations within a geometric system. Any matrix that represents a particular linear transformation reflects the properties of that transformation.

Since similar matrices share all the intrinsic geometric properties of a transformation, an important problem is to find a simple canonical form for each similarity class. This simple canonical form can be found by computing the eigenvalues and eigenvectors. Two methods for computing eigenvalues are illustrated, along with a method for finding eigenvectors.

The Characteristic Polynomial

The characteristic equation for a matrix can be written as

$$\begin{aligned} AX &= \lambda X \\ AX - \lambda X &= 0 \\ (A - \lambda I)X &= 0 \\ X &= 0 \quad \text{Trivial Solution} \\ \det(A - \lambda I) &= 0 \quad \text{Non-trivial Solution} \end{aligned}$$

Expansion of the non-trivial characteristic equation yields the characteristic polynomial

$$s_0\lambda^n + s_1\lambda^{n-1} + \cdots + s_{n-1}\lambda + s_n = 0.$$

The three programs below combine to determine the characteristic polynomial for an arbitrary matrix on the stack.

The first program, TRCN, creates a list of the traces of the first n powers of the matrix.

The second program, SYM, uses the list created by TRCN to compute the coefficients of the characteristic polynomial.

The final program, PSERS, uses the coefficients from SYM and a variable name entered into level 1 to create an expression of the characteristic polynomial.

Key in the first program.

```
[CLEAR]
«  DUP SIZE 1 GET → g
n «  g 'tmp' STO {} 1 n
START 0 1 n FOR i tmp i
DUP 2 →LIST GET + NEXT 1
→LIST + 'tmp' g STO*
NEXT 'tmp'
PURGE » » [ENTER] [<>]
```

```
1: « DUP SIZE 1.00 GET
   → g n « g 'tmp' STO
   { } 1.00 n START
   0.00 1.00 n FOR i
```

Store the program.

'TRCN [STO]

```
4:
3:
2:
1:
```

Key in the second program.

« DUP SIZE → b n «
{1} 1 n FOR i → s
« 0 1 i FOR j b j
GET s i j - 1 + GET ×
- NEXT i ÷ 1 →LIST s
SWAP + » NEXT » »
[ENTER] [<>]

```
1: « DUP SIZE → b n « {
   1.00 } 1.00 n FOR i
   → s « 0.00 1.00 i
   FOR j b j GET s i j
```

Store the program.

'SYM [STO]

```
4:
3:
2:
1:
```

Key in the final program.

« → x « LIST→ 0 SWAP
1 FOR n n 1 + ROLL
x n 1 - ^ × + -1
STEP » » [ENTER] [<>]

```
1: « → x « LIST→ 0.00
   SWAP 1.00 FOR n n
   1.00 + ROLL x n 1.00
   - ^ * + -1.00 STEP »
```

Store the program.

'PSERS [STO]

```
4:
3:
2:
1:
```

Find the characteristic polynomial for the following matrix.

$$ARR = \begin{bmatrix} -17 & -57 & -69 \\ 1 & 5 & 3 \\ 5 & 15 & 21 \end{bmatrix}$$

Key in the coefficient matrix.

[[-17 -57 -69[1 5 3
[5 15 21 [ENTER]

```
2:
1: [[ -17.00 -57.00 -6…
    [ 1.00 5.00 3.00 ]
    [ 5.00 15.00 21.00…
```

Create a list of the traces of the first n powers for the matrix.

[USER] TRCN

```
2:
1: { 9.00 41.00 225.00
   }
PSERS SYM TRCN UT
```

Compute the coefficients of the characteristic polynomial.

SYM

```
2:
1: { 1.00 -9.00 20.00
   -12.00 }
PSERS SYM TRCN UT
```

Create the algebraic expression of the characteristic polynomial with the variable name L.

'L' PSERS

```
3:
2:
1:    'L^3-9*L^2+20*L-12'
PSERS SYM TRCN UT
```

The characteristic polynomial is

$$\lambda^3 - 9\lambda^2 + 20\lambda - 12$$

Store the polynomial as the current expression in EQ for the following problem section.

[SOLV]
STEQ

```
3:
2:
1:
STEQ RCEQ SOLVR ISOL QUAD SHOW
```

Compute Eigenvalues from Expansion

The eigenvalues of a matrix can be found by solving for the roots of the characteristic polynomial.

Find the eigenvalues for the characteristic polynomial stored in the previous problem section.

Clear the stack and set the display mode to two decimal places.

[CLEAR]
[MODE] 2 FIX

Clear the current plot parameters.

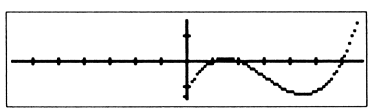
[PLOT]
'PPAR [PURGE]

Adjust the plot height by ten.

10 *H

Draw a plot of the characteristic polynomial, which was stored in EQ in the previous problem.

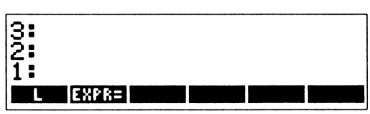
DRAW



Note the three roots of the quadratic indicate three distinct eigenvalues for the 3×3 matrix ARR.

Use the solver to set guesses for the roots and solve for the three eigenvalues.

[ATTN]
[SOLV]
SOLVR



Make an initial guess of 0.5 for the first root.

0.5 [L]

```
L: 0.50
2:
1:
 L  EXPR=
```

Solve for the first root.

Pressing the [ENTER] key below will display the intermediate values during calculation.

[ ] [L] [ENTER]

```
L: 1.00
Zero
1:                1.00
 L  EXPR=
```

The first eigenvalue $\lambda_1 = 1$.

Make an initial guess of 2.5 for the second eigenvalue.

[CLEAR]
2.5 [L]

```
L: 2.50
2:
1:
 L  EXPR=
```

Solve for the root.

[ ] [L] [ENTER]

```
L: 2.00
Zero
1:                2.00
 L  EXPR=
```

The second eigenvalue $\lambda_2 = 2$.

Make an initial guess of 5 for the third eigenvalue.

[CLEAR]
5 [L]

```
L: 5.00
2:
1:
 L  EXPR=
```

Solve for the root.

[ ] [L] [ENTER]

```
L: 6.00
Zero
1:                6.00
 L  EXPR=
```

The third eigenvalue $\lambda_3 = 6$.

Compute Eigenvectors

We can compute the eigenvectors corresponding to the three eigenvalues found in the previous problem.

$$ARR = \begin{bmatrix} -17 & -57 & -69 \\ 1 & 5 & 3 \\ 5 & 15 & 21 \end{bmatrix}$$

Clear the stack and set the display mode to one decimal place.

[CLEAR]
[MODE] 1 FIX

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the matrix ARR.

[[-17 -57 -69 [1 5 3
[5 15 21 [ENTER]

```
1: [[ -17.0 -57.0 -69....
    [ 1.0 5.0 3.0 ]
    [ 5.0 15.0 21.0 ]]
 STD [ FIX ] SCI  ENG [ DEG ] RAD
```

Create the 3×3 Identity matrix I.

3 [ENTER]

```
3:
2: [[ -17.0 -57.0 -69....
1:                    3.0
 STD [ FIX ] SCI  ENG [ DEG ] RAD
```

[ARRAY] IDN

```
1: [[ 1.0 0.0 0.0 ]
    [ 0.0 1.0 0.0 ]
    [ 0.0 0.0 1.0 ]]
 SIZE  RDM  TRN  CON  IDN  RSD
```

Form λ*I for $\lambda_1 = 1$.

1 [ENTER]
[×]

```
1: [[ 1.0 0.0 0.0 ]
    [ 0.0 1.0 0.0 ]
    [ 0.0 0.0 1.0 ]]
 SIZE  RDM  TRN  CON  IDN  RSD
```

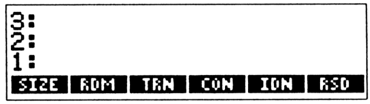
Subtract from ARR to obtain the matrix $(ARR - \lambda_1 I)$.

[-]

```
1: [[ -18.0 -57.0 -69....
    [ 1.0 4.0 3.0 ]
    [ 5.0 15.0 20.0 ]]
 SIZE  RDM  TRN  CON  IDN  RSD
```

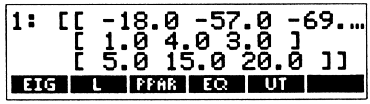
Store the matrix $(ARR - \lambda_1 I)$ as EIG.

'EIG [STO]



Recall the matrix EIG.

[USER] EIG



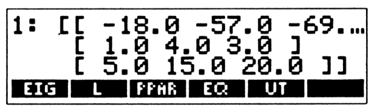
Verify that $\det(A - \lambda I) = 0$.

[ARRAY]
DET

The determinant is approximately zero.

Recall matrix EIG once more.

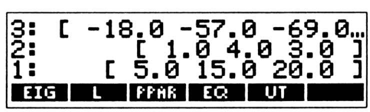
[DROP]
[USER] EIG



Reduce to row echelon form to solve for the eigenvector X_1, where $(A - \lambda_1 I)X_1 = 0$.

Enter [EDIT] mode and use the [DEL] key to remove the outer array brackets and form three individual row vectors. Each row vector corresponds to one equation of the system. After the edit, the row vectors can be [ENTER]ed:

```
[ -18 -57 -69 ]
 [ 1 4 3 ]
 [ 5 15 20 ] [ENTER]
```



Note the utilities in the section entitled "Orthogonality" which can also perform the modification of the form of the matrix above.

Use the program UT, described in the problem section "Homogeneous System", to reduce the matrix to upper triangular form.

UT

```
3: [ -18.0 -57.0 -69.0…
2:      [ 0.0 0.8 -0.8 ]
1: [ 0.0 0.0 -1.0E-11 ]
EIG | L | PPAR | EQ | UT |
```

Remove the vector that represents the equation 0 = 0.

DROP

```
3:
2: [ -18.0 -57.0 -69.0…
1:      [ 0.0 0.8 -0.8 ]
EIG | L | PPAR | EQ | UT |
```

Enter the equation represented by second vector.

'.8 × X2 - .8 × X3 =0
ENTER

```
3: [ -18.0 -57.0 -69.0…
2:      [ 0.0 0.8 -0.8 ]
1:      '0.8*X2-0.8*X3=0'
EIG | L | PPAR | EQ | UT |
```

Solve for x_2.

ALGEBRA
'X2 ENTER

```
3:      [ 0.0 0.8 -0.8 ]
2:      '0.8*X2-0.8*X3=0'
1:                  'X2'
COLCT | EXPAN | SIZE | FORM | OBSUB | EXSUB
```

Isolate the term.

ISOL

```
3: [ -18.0 -57.0 -69.0…
2:      [ 0.0 0.8 -0.8 ]
1:           '0.8*X3/0.8'
TAYLR | ISOL | QUAD | SHOW | OBGET | EXGET
```

Collect terms.

COLCT

```
3: [ -18.0 -57.0 -69.0…
2:      [ 0.0 0.8 -0.8 ]
1:                  'X3'
COLCT | EXPAN | SIZE | FORM | OBSUB | EXSUB
```

We obtain $x_2=x_3$. Remove this solution and the second vector from the stack.

DROP
DROP

```
3:
2:
1: [ -18.0 -57.0 -69.0…
COLCT | EXPAN | SIZE | FORM | OBSUB | EXSUB
```

Enter the equation represented by the first vector, substituting x_3 with x_2.

'-18 × X1 -57 × X2 -69 × X2 =0 ENTER

```
2: [ -18.0 -57.0 -69.0…
1: '-18*X1-57*X2-69*X2=
   0'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Solve for x_1.

'X1 ENTER

```
3: [ -18.0 -57.0 -69.0…
2: '-18*X1-57*X2-69*X2…
1:                 'X1'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Isolate the term.

ISOL

```
2: [ -18.0 -57.0 -69.0…
1: '(69*X2+57*X2)/(-18)
   '
TAYLR ISOL QUAD SHOW OBGET EXGET
```

Collect like terms.

COLCT

```
3:
2: [ -18.0 -57.0 -69.0…
1:          '-(7.0*X2)'
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

We get $x_1 = -7*x_2$.

Therefore a solution eigenvector is $x_1 = -7, x_2 = 1, x_3 = 1$, or $X_1 = [-7\ 1\ 1]$. Verify that $(A - \lambda I)X = 0$.

CLEAR
[-7 1 1 ENTER

```
3:
2:
1:      [ -7.0 1.0 1.0 ]
COLCT EXPAN SIZE FORM OBSUB EXSUB
```

Recall $(A - \lambda I)$.

USER
EIG

```
1: [[ -18.0 -57.0 -69.…
    [ 1.0 4.0 3.0 ]
    [ 5.0 15.0 20.0 ]]
EIG L PPAR EQ UT
```

Multiply the two matrices.

SWAP
×

```
3:
2:
1:       [ 0.0 0.0 0.0 ]
EIG L PPAR EQ UT
```

The result is 0, verifying that X_1 is indeed an eigenvector associated with λ_1.

The same procedure can be followed to find eigenvectors for $\lambda_2 = 2$ and $\lambda_3 = 6$.

Purge the user variables and programs used in the last three sections.
`{ 'EIG' 'L' 'PPAR' 'EQ' 'UT' }` [PURGE].

Compute Eigenvalues from $|\lambda I - A|$

Find eigenvalues directly from the function det($\lambda I - A$), without computing the characteristic polynomial.

$$A = \begin{bmatrix} -7.8 & -29.7 & -39.6 \\ 0 & 2.1 & 0 \\ 3.3 & 9.9 & 15.3 \end{bmatrix}$$

Clear the stack and set the display mode to two decimal places.

[CLEAR]
[MODE] 2 FIX

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Clear the current plot parameters.

'PPAR [PURGE]

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the 3×3 matrix.

```
[[-7.8 -29.7 -39.6
[0 2.1 0[3.3 9.9 15.3
```
[ENTER]

```
1: [[ -7.80 -29.70 -39...
    [ 0.00 2.10 0.00 ]
    [ 3.30 9.90 15.30 ...
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Store matrix A.

'A [STO]

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Enter a program that computes the function det($\lambda I - A$), with λ the independent variable.

« 3 IDN L × A - DET »
[ENTER]

```
2:
1: « 3.00 IDN L * A -
   DET »
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Store the function as the current expression in EQ.

[PLOT]
STEQ

```
3:
2:
1:
 STEQ RCEQ PMIN PMAX INDEP DRAW
```

Adjust the plot height.

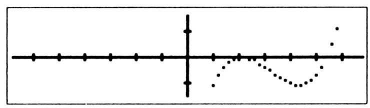
5 *H

Set a larger resolution.

2 RES

Plot the function, using λ for the abscissa. The program takes several minutes to complete, as it computes the determinant for each point plotted.

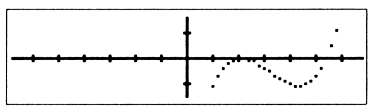
DRAW



The curve shows that there are only two distinct roots. The leftmost root, which is a local maximum, must represent a double eigenvalue root.

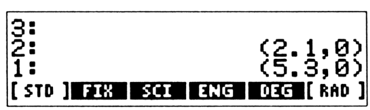
Digitize the roots to set initial guesses for the root solver.

[>] . . . [>] [INS]
[>] . . . [>] [INS]



Set the standard display mode.

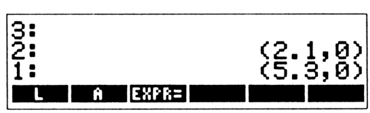
[ATTN]
[MODE] STD



NOTE: The values displayed will vary by differences in the digitizing position from the graphics display.

Use the Solver to find the roots of the curve.

[SOLV]
SOLVR



Solve for the rightmost root.

L ☐ L [ENTER]

One root is $\lambda_1 = 5.40$.

Drop this result from the stack and solve for the next root.

[DROP]
L ☐ L [ENTER]

The double eigenvalue is $\lambda_2 = \lambda_3 = 2.10$.

Least Squares

The method of least squares is a standard statistical algorithm used to fit a curve to data in order to estimate a function, predict a trend, or approximate missing data values. Least squares results can easily be calculated on the HP-28C and the graphic display is particularly useful for examining the fit to the original data.

Straight Line Fitting

Find the least squares straight line fit to the four points: (0,1), (1,3), (2,4), and (3,4).

The least squares solution is given by $Y = MV$ to fit the line $y = ax + b$.

NOTE: The solution provided below serves to illustrate matrix operations, and could be replaced, in the case of $y = ax + b$, with the statistical functions (Linear Regression) of the HP-28C.

$$Y = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 4 \end{bmatrix}$$

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix}$$

$$V = \begin{bmatrix} a \\ b \end{bmatrix}$$

Solving for V gives us

$$V = \frac{M^T Y}{M^T M}$$

CLEAR
MODE 2 FIX

```
3:
2:
1:
STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the y values of the data points.

`[[1[3[4[4` ENTER

```
1: [[ 1.00 ]
    [ 3.00 ]
    [ 4.00 ]
STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Store the 4×1 matrix Y.

'Y [STO]

```
3:
2:
1:
STD [ FIX ] SCI ENG DEG [ RAD ]
```

Key in the a and b values representing the line $y = ax + b$.

[[0 1[1 1[2 1[3 1 [ENTER]

```
1: [[ 0.00 1.00 ]
    [ 1.00 1.00 ]
    [ 2.00 1.00 ]
STD [ FIX ] SCI ENG DEG [ RAD ]
```

Store the 4×2 matrix M.

'M [STO]

```
3:
2:
1:
STD [ FIX ] SCI ENG DEG [ RAD ]
```

Compute V using the least squares fitting method.

M [ENTER]
[ARRAY]
TRN
Y [×]

```
2:
1: [[ 23.00 ]
    [ 12.00 ]]
SIZE RDM TRN CON IDN RSD
```

M [ENTER]
TRN
M [×]

```
2: [[ 23.00 ]  [ 12.00…
1: [[ 14.00 6.00 ]
    [ 6.00 4.00 ]]
SIZE RDM TRN CON IDN RSD
```

[÷]

```
2:
1: [[ 1.00 ]
    [ 1.50 ]]
SIZE RDM TRN CON IDN RSD
```

Store the coefficients from matrix V in the individual variables a and b.

ARRY→

```
3:                   1.00
2:                   1.50
1:         { 2.00 1.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

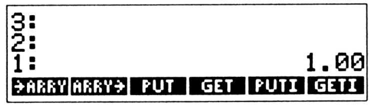
Drop the dimension list.

[DROP]

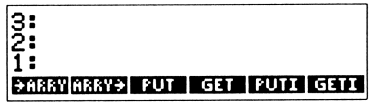
```
3:
2:                   1.00
1:                   1.50
→ARRY ARRY→ PUT GET PUTI GETI
```

Store the two coefficients.

'B [STO]

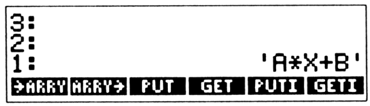


'A [STO]



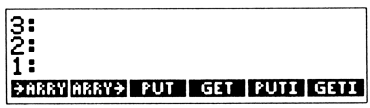
Enter the equation for the straight line.

'A × X + B [ENTER]



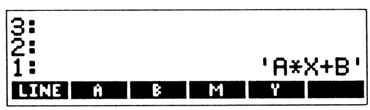
Store the equation.

'LINE [STO]



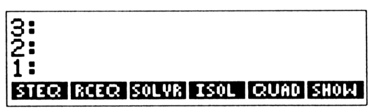
Recall equation LINE.

[USER] ≣ LINE ≣



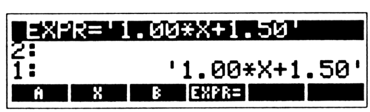
Store the line equation as the current expression in EQ.

[SOLV] ≣ STEQ ≣



Use the Solver to compute the desired line.

≣ SOLVR ≣ ≣ EXPR= ≣



The straight line fit to the data is the equation $y = 1.5x + 1$.

Now use the PLOT menu to draw the line and verify the fit to the data.

Clear the current plot parameters.

PLOT
'PPAR PURGE

```
3:
2:
1:              '1.00*X+1.50'
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Establish X as the independent variable.

'X INDEP

```
3:
2:
1:              '1.00*X+1.50'
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Adjust the height by 5.

5 *H

```
3:
2:
1:              '1.00*X+1.50'
PPAR RES AXES CENTR *W *H
```

Recenter the axes so that the point (0,1) can be viewed on the plot.

(-1,-1) ENTER
AXES

```
3:
2:
1:              '1.00*X+1.50'
PPAR RES AXES CENTR *W *H
```

Now move to the Statistics menu to set up a scatter plot.

STAT CLΣ

```
3:
2:
1:              '1.00*X+1.50'
Σ+ Σ- NΣ CLΣ STOΣ RCLΣ
```

Enter the four data points into ΣDAT.

[0,1 Σ+
[1,3 Σ+
[2,4 Σ+
[3,4 Σ+

```
3:
2:
1:              '1.00*X+1.50'
Σ+ Σ- NΣ CLΣ STOΣ RCLΣ
```

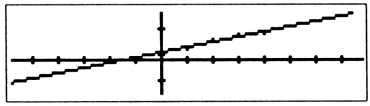
Enter a program that will overlay the function plot onto the scatter plot.

PLOT
« CLLCD DRWΣ DRAW
ENTER

```
3:
2:              '1.00*X+1.50'
1:   « CLLCD DRWΣ DRAW »
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Draw the plot.

[EVAL]



We can see from the plot that the line fits the four points well.

Purge the variables used in the problem section. { 'ΣPAR' 'ΣDAT' 'PPAR' 'EQ' 'A' 'B' 'M' 'Y'} [PURGE].

Quadratic Polynomial

According to Newton's Second Law of Motion, a body near the earth's surface falls vertically downward according to the equation

$$y = y_0 + v_0 t + \frac{1}{2} g\, t^2$$

where

y = vertical displacement at time t.
y_0 = initial vertical displacement at time $t_0 = 0$.
v_0 = initial velocity at time $t_0 = 0$.
g = Newton's constant of acceleration of gravity near the earth's surface.

An experiment is performed to evaluate g. A weight is released with unknown initial displacement and velocity. At a fixed time interval the distance fallen from a fixed reference point is measured. The following results are obtained: At times t = .1, .2, ...5 seconds the weight has fallen y = −.055, .094, .314, .756, and 1.138 meters, respectively, from the reference point. Calculate the value for Newton's constant g using these data.

We will fit the quadratic curve

$$y = a + bt + ct^2$$

to the five data points. The least squares solution is given by

$$Y = MV$$

where

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \\ 1 & t_4 & t_4^2 \\ 1 & t_5 & t_5^2 \end{bmatrix}$$

and

$$V = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Solving for V gives us

$$V = \frac{M^T Y}{M^T M}$$

Clear the stack and set the display mode to three decimal places.

CLEAR
MODE 3 FIX

```
3:
2:
1:
 STO [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the matrix of y values.

```
[[-.055[.094[.314[.756[
1.138 ENTER
```

```
1: [[ -0.055 ]
    [ 0.094 ]
    [ 0.314 ]
 STO [ FIX ] SCI  ENG  DEG [ RAD ]
```

Store the 5×1 matrix Y.

'Y STO

```
3:
2:
1:
 STO [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the components of array M.

Enter $row_1 = 1, .1, .1^2$.

1 ENTER
.1 ENTER
ENTER
x^2

```
3:                           1.000
2:                           0.100
1:                           0.010
 STO [ FIX ] SCI  ENG  DEG [ RAD ]
```

Enter $row_2 = 1, .2, .2^2$.

1 [ENTER]
.2 [ENTER]
[ENTER]
[x^2]

```
3:                       1.000
2:                       0.200
1:                       0.040
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Enter $row_3 = 1, .3, .3^2$.

1 [ENTER]
.3 [ENTER]
[ENTER]
[x^2]

```
3:                       1.000
2:                       0.300
1:                       0.090
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Enter $row_4 = 1, .4, .4^2$.

1 [ENTER]
.4 [ENTER]
[ENTER]
[x^2]

```
3:                       1.000
2:                       0.400
1:                       0.160
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Finally, enter $row_5 = 1, .5, .5^2$.

1 [ENTER]
.5 [ENTER]
[ENTER]
[x^2]

```
3:                       1.000
2:                       0.500
1:                       0.250
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Key in the dimension of *M*.

{5 3 [ENTER]

```
3:                       0.500
2:                       0.250
1:           { 5.000 3.000 }
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Put the components into the array.

[ARRAY]
→ARRY

```
1: [[ 1.000 0.100 0.01…
    [ 1.000 0.200 0.04…
    [ 1.000 0.300 0.09…
 →ARRY ARRY→ PUT  GET  PUTI GETI
```

Store matrix *M*.

'M [STO]

```
3:
2:
1:
 →ARRY ARRY→ PUT  GET  PUTI GETI
```

Compute V using the least squares method.

M [ENTER]
TRN
Y [×]

```
1: [[ 2.247 ]
    [ 0.979 ]
    [ 0.437 ]]
SIZE RDM TRN CON IDN RSD
```

M [ENTER]
TRN
M [×]
[÷]

```
1: [[ -0.121 ]
    [ 0.099 ]
    [ 4.914 ]]
SIZE RDM TRN CON IDN RSD
```

Store matrix V.

'V [STO]

```
3:
2:
1:
SIZE RDM TRN CON IDN RSD
```

Evaluate g, Newton's constant of gravity. Get element c from the solution vector V, then multiply c by 2. $g = 2*c$.

V [ENTER]
{3 1}
GET
2 [×]

```
3:
2:
1:                 9.829
→ARRY ARRY→ PUT GET PUTI GETI
```

Convert from m/sec^2 to ft/sec^2.

[LC] m [ENTER]
[LC] ft [ENTER]

```
3:                 9.829
2:                   'm'
1:                  'ft'
→ARRY ARRY→ PUT GET PUTI GETI
```

[CONVERT]

```
3:
2:                32.246
1:                  'ft'
→ARRY ARRY→ PUT GET PUTI GETI
```

The result is $g = 32.246$ ft/sec^2.

Now use the solver to compute the desired quadratic polynomial.

'A + B × T + C × T^2
[ENTER]

```
3:                32.246
2:                  'ft'
1:       'A+B*T+C*T^2'
→ARRY ARRY→ PUT GET PUTI GETI
```

Store equation POLY.

'POLY [STO]

```
3:
2:                32.246
1:                 'ft'
→ARRY ARRY→ PUT GET PUTI GETI
```

Get the coefficients from matrix V.

V [ENTER]

```
1: [[ -0.121 ]
    [ 0.099 ]
    [ 4.914 ]]
→ARRY ARRY→ PUT GET PUTI GETI
```

ARRY→

```
3:                 0.099
2:                 4.914
1:       { 3.000 1.000 }
→ARRY ARRY→ PUT GET PUTI GETI
```

Drop the dimension list.

[DROP]

```
3:                -0.121
2:                 0.099
1:                 4.914
→ARRY ARRY→ PUT GET PUTI GETI
```

Store the three coefficients a, b, and c.

'C [STO]

```
3:                  'ft'
2:                -0.121
1:                 0.099
→ARRY ARRY→ PUT GET PUTI GETI
```

'B [STO]

```
3:                32.246
2:                  'ft'
1:                -0.121
→ARRY ARRY→ PUT GET PUTI GETI
```

'A [STO]

```
3:
2:                32.246
1:                  'ft'
→ARRY ARRY→ PUT GET PUTI GETI
```

Recall the equation.

[USER] POLY

```
3:                32.246
2:                  'ft'
1:        'A+B*T+C*T^2'
A B C POLY V M
```

Store the equation as the current expression EQ.

```
SOLV
STEQ
```

```
3:
2:                32.246
1:                  'ft'
STEQ RCEQ SOLVR ISOL QUAD SHOW
```

Use the Solver to compute the desired equation.

```
SOLVR
EXPR=
```

```
EXPR='-0.121+0.099*T+
1: '-0.121+0.099*T+
  4.914*T^2'
A B T C EXPR=
```

The least squares solution equation is $-0.121+0.099t+4.914t^2$.

Next, we will overlay the function curve over a scatter plot of the data points to verify the fit.

First, clear the current plot parameters and establish t as the independent variable.

```
CLEAR
PLOT 'PPAR PURGE
'T INDEP
```

```
3:
2:
1:
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Adjust the plot width by .1, to plot 0.1 second intervals along the abscissa.

```
.1 *W
```

```
3:
2:
1:
PPAR RES AXES CENTR *W *H
```

Next use the Statistics menu to create the scatter plot.

```
STAT
CLΣ
```

```
3:
2:
1:
Σ+ Σ- NΣ CLΣ STOΣ RCLΣ
```

Enter the data points for the scatter plot.

```
[.1 -.055 Σ+
[.2 .094 Σ+
[.3 .314 Σ+
[.4 .756 Σ+
[.5 1.138 Σ+
```

```
3:
2:
1:
Σ+ Σ- NΣ CLΣ STOΣ RCLΣ
```

Now write a program to overlay the two plots.

[PLOT]
« CLLCD DRWΣ DRAW
[ENTER]

```
3:
2:
1:   « CLLCD DRWΣ DRAW »
STEQ RCEQ PMIN PMAX INDEP DRAW
```

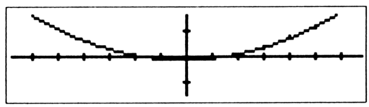
Store program PLT.

'PLT [STO]

```
3:
2:
1:
STEQ RCEQ PMIN PMAX INDEP DRAW
```

Draw the plot.

[USER] PLT

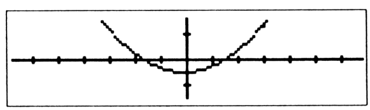


You may wish to rescale the plot height to obtain a better view of the fit of the first two data points.

.25 *H

```
3:
2:
1:
PPAR RES AXES CENTR *W *H
```

[USER]
PLT



The plots show a good fit of the quadratic polynomial to the five data points.

Markov Chains

A Markov Chain is a system that moves from state to state, and in which the probability of transition to a next state depends only on the preceding state. The system states can be predicted at particular points in time using transition probabilities.

The transition matrix for the Markov Process is the $n \times n$ matrix $P = [p_{ij}]$ where p_{ij} = probability of transition directly from state j to state i, and $\sum_{i=1}^{n} p_{ij} = 1$.

The components of the state vector $X^{(n)}$ signify the probability that the system is in state i at the n^{th} observation.

$$X^{(n)} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}$$

The model for the system is described by $X^{(n+1)} = P\,X^{(n)}$, where the transition matrix applied to the current state determines the next state.

Steady State of a System

A chemist runs an experiment where colored films are immersed in a solution for a brief time period, resulting in a possible color change. He calculates the color changes according to the following probabilities.

Original Color			New Color
Magenta	Cyan	Yellow	
.8	.3	.2	Magenta
.1	.2	.6	Cyan
.1	.5	.2	Yellow

Determine to two decimal places the probable future color of a cyan film dipped in the solution several times.

[CLEAR]
[MODE] 2 FIX

```
3:
2:
1:
STD [ FIX ] SCI ENG DEG [ RAD ]
```

Key in the 3×3 transition matrix P.

`[[.8 .3 .2[.1 .2 .6[.1 .5 .2` [ENTER]

```
1: [[ 0.80 0.30 0.20 ]
    [ 0.10 0.20 0.60 ]
    [ 0.10 0.50 0.20 ]]
STD [ FIX ] SCI ENG DEG [ RAD ]
```

`'P` [STO]

```
3:
2:
1:
STD [ FIX ] SCI ENG DEG [ RAD ]
```

Key in the initial state vector X^0. This vector represent an initial state of cyan.

`[[0[1[0` [ENTER]

```
1: [[ 0.00 ]
    [ 1.00 ]
    [ 0.00 ]]
STD [ FIX ] SCI ENG DEG [ RAD ]
```

`'X` [STO]

```
3:
2:
1:
STD [ FIX ] SCI ENG DEG [ RAD ]
```

Key in the initial value for n = current state.

0 [ENTER]
'N [STO]

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Write a program to compute the next future state.

« N 1 + 'N' STO P
SWAP × » [ENTER]

```
2:
1: « N 1.00 + 'N' STO P
   SWAP * »
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Store program MARK.

'MARK [STO]

```
3:
2:
1:
 STD [ FIX ] SCI  ENG  DEG [ RAD ]
```

Recall the initial state vector.

[USER]
X

```
1: [[ 0.00 ]
    [ 1.00 ]
    [ 0.00 ]]
 MARK  N  X  P
```

Compute the next state.

MARK

```
1: [[ 0.30 ]
    [ 0.20 ]
    [ 0.50 ]]
 MARK  N  X  P
```

After one observation, the color is most likely to be yellow. Compute the next state.

MARK

```
1: [[ 0.40 ]
    [ 0.37 ]
    [ 0.23 ]]
 MARK  N  X  P
```

After two observations, the color is most likely to be either magenta or cyan. Continue computing future states until a final steady state is reached.

MARK

```
1: [[ 0.48 ]
    [ 0.25 ]
    [ 0.27 ]]
 MARK  N  X  P
```

MARK

```
1: [[ 0.51 ]
    [ 0.26 ]
    [ 0.23 ]]
MARK N X P
```

MARK

```
1: [[ 0.53 ]
    [ 0.24 ]
    [ 0.23 ]]
MARK N X P
```

MARK

```
1: [[ 0.54 ]
    [ 0.24 ]
    [ 0.22 ]]
MARK N X P
```

MARK

```
1: [[ 0.55 ]
    [ 0.23 ]
    [ 0.22 ]]
MARK N X P
```

MARK

```
1: [[ 0.55 ]
    [ 0.23 ]
    [ 0.21 ]]
MARK N X P
```

MARK

```
1: [[ 0.56 ]
    [ 0.23 ]
    [ 0.21 ]]
MARK N X P
```

MARK

```
1: [[ 0.56 ]
    [ 0.23 ]
    [ 0.21 ]]
MARK N X P
```

The system has reached a steady state. Determine how many observations were completed to reach this final state.

N

```
3:
2: [[ 0.56 ]  [ 0.23 ]…
1:                 10.00
MARK N X P
```

The system reaches a steady state after $n = 10$ observations. The probable future color of an initially cyan film immersed several times is .56 magenta, .23 cyan, and .21 yellow.

Purge the variables used in this problem section. `{ 'MARK' 'N' 'X' 'P' }` [PURGE].

An Example:

Matrix manipulations are used to solve complex, multi-dimensional problems. The following sections illustrate use of the HP-28C matrix capabilities in a market with challenging economic issues. These same analytical tools can be applied across many industries.

Forest Management

When a forest is managed by a sustainable harvesting policy, every tree harvested is replaced by a new seedling, so the total population quantity remains constant. A matrix model can be developed to assist in determining optimal harvesting procedures. The model is based on categorizing the trees into height/price classes and computing an optimal sustainable yield for a long-range time period.

The Sustainable Harvesting Cycle is represented by:

Forest ready for harvest − harvest + new seedlings = forest after harvest, or

$$GX - Y + RY = X$$

where

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

X = Nonharvest vector, the trees that remain after the harvest and replanting.

x_i = number of trees in the i th class.

i ranges from 1 to n, where there are n height/price classes.

$S = \sum_{i=1}^{n} x_i$ = total number of trees sustained.

Tree growth between harvests is designated by g_i, the fraction of trees that grow from class i to class $i+1$.

$1 - g_i$ = fraction of trees that remain in class i.

The growth matrix is

$$G = \begin{bmatrix} 1-g_1 & 0 & 0 & \cdot & 0 \\ g_1 & 1-g_2 & 0 & \cdot & 0 \\ 0 & g_2 & 1-g_3 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & 1-g_{n-1} & 0 \\ 0 & 0 & 0 & g_{n-1} & 1 \end{bmatrix}$$

GX = Nonharvest vector after growth period, or forest ready for harvest.

$$Y = \begin{bmatrix} y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{bmatrix}$$

Y = Harvest vector, or trees removed at harvest.

$$R = \begin{bmatrix} 1 & 1 & 1 & \cdot & \cdot & 1 \\ 0 & 0 & 0 & \cdot & \cdot & 0 \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

R = Replacement matrix.

RY = New seedling vector, or trees planted after harvest.

The Harvest Model

A harvester has a crop of 120 silver fir trees to sell annually for Christmas trees. After last year's harvest, his forest had the following configuration.

Class (i)	**Height interval in feet** (h_i)	**Number of trees** (x_i)
1	[0,4)	15
2	[4,8)	20
3	[8,12)	35
4	[12,16)	30
5	[16,∞)	20

During the growth period, 6 trees in class 1 grew to the next height class, as did 13 trees in class 2, 10 trees in class 3, and 4 trees in class 4. If he sustainably harvests 8 trees of class 2, 6 trees of class 3, 13 trees of class 4, and 6 trees of class 5, what is the configuration of his crop after harvest and replanting?

CLEAR
MODE 2 FIX

```
3:
2:
1:
STO [ FIX ] SCI ENG DEG [ RAD ]
```

Enter the 5×1 nonharvest vector X.

[[15[20[35[30[20 ENTER

```
1: [[ 15.00 ]
    [ 20.00 ]
    [ 35.00 ]
STO [ FIX ] SCI ENG DEG [ RAD ]
```

'X STO

```
3:
2:
1:
STO [ FIX ] SCI ENG DEG [ RAD ]
```

Compute the growth fractions for each height class. First, compute $g_1 = 6/x_1$.

6 ENTER
15 ÷

```
3:
2:
1:                  0.40
STO [ FIX ] SCI ENG DEG [ RAD ]
```

'G1 [STO]

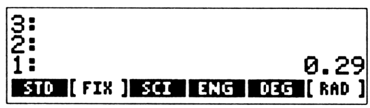
Compute $g_2 = 13/x_2$.

13 [ENTER]
20 [÷]

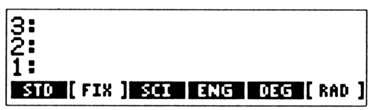
'G2 [STO]

Compute $g_3 = 10/x_3$.

10 [ENTER]
35 [÷]

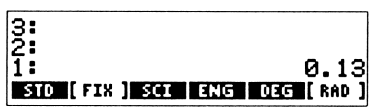


'G3 [STO]

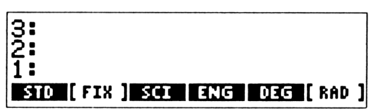


Compute $g_4 = 4/x_4$.

4 [ENTER]
30 [÷]



'G4 [STO]



Enter the 5×5 growth matrix G.

Enter row_1.

USER
1 ENTER
G1
−
0 ENTER
ENTER
ENTER
ENTER

```
3:                    0.00
2:                    0.00
1:                    0.00
 G4 | G3 | G2 | G1 | X |
```

Enter row_2.

G1
1 ENTER
G2
−
0 ENTER
ENTER
ENTER

```
3:                    0.00
2:                    0.00
1:                    0.00
 G4 | G3 | G2 | G1 | X |
```

Enter row_3.

0 ENTER
G2
1 ENTER
G3
−
0 ENTER
ENTER

```
3:                    0.71
2:                    0.00
1:                    0.00
 G4 | G3 | G2 | G1 | X |
```

Enter row_4.

0 ENTER
ENTER
G3
1 ENTER
G4
−
0 ENTER

```
3:                    0.29
2:                    0.87
1:                    0.00
 G4 | G3 | G2 | G1 | X |
```

Enter row_5.

0 [ENTER]
[ENTER]
[ENTER]
G4
1 [ENTER]

```
3:                  0.00
2:                  0.13
1:                  1.00
 G4 | G3 | G2 | G1 | X |
```

Enter the dimensions of G.

{5 5} [ENTER]

```
3:                  0.13
2:                  1.00
1:       { 5.00 5.00 }
 G4 | G3 | G2 | G1 | X |
```

Store matrix G.

[ARRAY]
→ARRY

```
1:  [[ 0.60 0.00 0.00 0…
     [ 0.40 0.35 0.00 0…
     [ 0.00 0.65 0.71 0…
→ARRY|ARRY→| PUT | GET | PUTI | GETI
```

'G [STO]

```
3:
2:
1:
→ARRY|ARRY→| PUT | GET | PUTI | GETI
```

Enter the 5×1 harvest vector Y.

[[0[8[6[13[6 [ENTER]

```
1:  [[ 0.00 ]
     [ 8.00 ]
     [ 6.00 ]
→ARRY|ARRY→| PUT | GET | PUTI | GETI
```

'Y [STO]

```
3:
2:
1:
→ARRY|ARRY→| PUT | GET | PUTI | GETI
```

Create the replacement matrix R. First enter the dimensions of R.

{5 5} [ENTER]

```
3:
2:
1:       { 5.00 5.00 }
→ARRY|ARRY→| PUT | GET | PUTI | GETI
```

Create a constant matrix whose entries are all zero.

0 [ENTER]
[CON]

```
1: [[ 0.00 0.00 0.00 0...
    [ 0.00 0.00 0.00 0...
    [ 0.00 0.00 0.00 0...
SIZE RDM TRN CON IDN RSD
```

Now enter 1 across the entire first row of *R* .

{1 1} [ENTER]

```
3:
2: [[ 0.00 0.00 0.00 0...
1:        { 1.00 1.00 }
SIZE RDM TRN CON IDN RSD
```

1 [PUTI]
1 [PUTI]
1 [PUTI]
1 [PUTI]
1 [PUTI]

```
3:
2: [[ 1.00 1.00 1.00 1...
1:        { 2.00 1.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

Drop the index list.

[DROP]

```
1: [[ 1.00 1.00 1.00 1...
    [ 0.00 0.00 0.00 0...
    [ 0.00 0.00 0.00 0...
→ARRY ARRY→ PUT GET PUTI GETI
```

Store matrix R.

'R [STO]

```
3:
2:
1:
→ARRY ARRY→ PUT GET PUTI GETI
```

Write a program to compute the configuration of the forest after harvest.

[USER]
« G X × Y - R Y × + »
[ENTER]

```
2:
1: « G X * Y - R Y * +
   »
R Y G G4 G3 G2
```

'CROP [STO]

```
3:
2:
1:
CROP R Y G G4 G3
```

Compute the new nonharvest vector with program CROP.

CROP

```
1: [[ 42.00 ]
    [ 5.00 ]
    [ 32.00 ]
CROP | R | Y | G | G4 | G3
```

Use [EDIT] or [VIEW↓] to view the entire vector. The [ATTN] key will exit EDIT mode.

The new nonharvest vector is

$$X = \begin{bmatrix} 42 \\ 5 \\ 32 \\ 23 \\ 18 \end{bmatrix}$$

The program can be used with the new nonharvest vector to predict new forest configurations using the same harvesting cycle annually.

Optimal Yield

If the harvester wishes to optimize his profit year after year, he must determine the optimal sustainable yield. This is achieved by harvesting all of the trees from one particular height/price class and no trees from any other class. The sustainable yield is thus a function of both price and growth rate, but independent of the current nonharvest vector. Note that if class k provides the maximum yield, the first year all classes $\geq k$ are harvested. In the following years only class k is harvested, and no trees will ever be present in higher classes.

S = total number of trees sustained in the forest.

$$P = \begin{bmatrix} p_1 & 0 & \cdot & 0 \\ \cdot & p_2 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & p_n \end{bmatrix} = \text{Price matrix}$$

p_i = price attained for class i.

$$GG = \begin{bmatrix} gg_1 \\ gg_2 \\ \cdot \\ \cdot \\ gg_n \end{bmatrix}$$

GG = Growth ratio matrix.

where

$$\begin{cases} gg_i = \dfrac{1}{\displaystyle\sum_{k=1}^{i-1} \frac{1}{g_k}} & \textit{for } i = 2 \ldots n \\ gg_1 = 0 \end{cases}$$

$$YL = \begin{bmatrix} yl_1 \\ yl_2 \\ \cdot \\ \cdot \\ yl_n \end{bmatrix}$$

YL = Yield vector.

yl_k = yield (total dollar amount) obtained by harvesting all of class i and no other class.

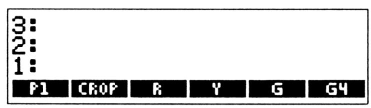
The optimal class to harvest can be selected by finding the maximum yl_k from yield vector YL, where

$$YL = P * S * GG$$

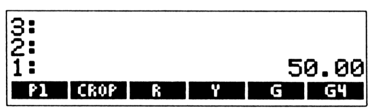
Suppose the market prices for the five classes are p_1 = \$0, p_2 = \$50, p_3 = \$100, p_4 = \$150, and p_5 = \$200. Determine which height class should be harvested.

Enter the market prices for the five classes and store in variables p_1 through p_5.

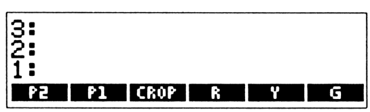
[CLEAR] [USER]
0 [ENTER]
'P1 [STO]



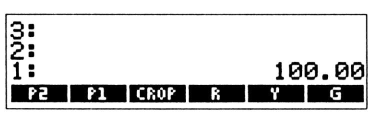
50 [ENTER]



'P2 [STO]

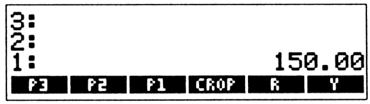


P2
2 [×]

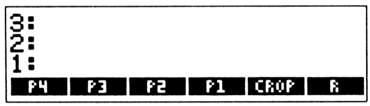


'P3 [STO]

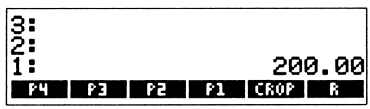
P2
3 [×]



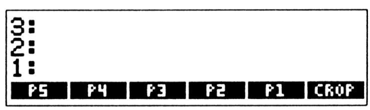
'P4 [STO]



P2
4 [×]

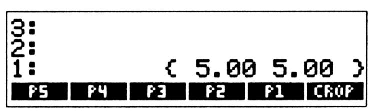


'P5 [STO]



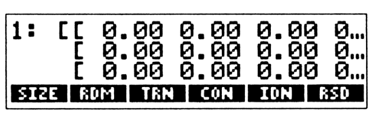
Enter the dimensions of P.

{5 5} [ENTER]



Create the 5×5 price matrix P. Since P is a sparse matrix, with most entries equal to zero, first create a constant array whose entries are all zero.

0 [ENTER]
[ARRAY] CON



Now enter the values p_i along the diagonal entries.

{1 1} [ENTER]

```
3:
2: [[ 0.00 0.00 0.00 0...
1:           { 1.00 1.00 }
SIZE RDM TRN CON IDN RSD
```

P1 [ENTER]

```
3: [[ 0.00 0.00 0.00 0...
2:           { 1.00 1.00 }
1:                    0.00
SIZE RDM TRN CON IDN RSD
```

PUTI

```
3:
2: [[ 0.00 0.00 0.00 0...
1:           { 1.00 2.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

Use the [EDIT] function to modify the displayed position index. The modified position index is then [ENTER]ed. Alternatively, you may [DROP] {1.00 2.00} from above and enter the position index {2 2}.

{ 2 2 } [ENTER]

```
3:
2: [[ 0.00 0.00 0.00 0...
1:           { 2.00 2.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

P2 [ENTER]

```
3: [[ 0.00 0.00 0.00 0...
2:           { 2.00 2.00 }
1:                   50.00
→ARRY ARRY→ PUT GET PUTI GETI
```

PUTI

```
3:
2: [[ 0.00 0.00 0.00 0...
1:           { 2.00 3.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

Use the [EDIT] function to modify the position index. The modified position index is then [ENTER]ed:

{ 3 3 } [ENTER]

```
3:
2: [[ 0.00 0.00 0.00 0...
1:           { 3.00 3.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

P3 [ENTER]

```
3: [[ 0.00 0.00 0.00 0...
2:          { 3.00 3.00 }
1:               100.00
→ARRY ARRY→ PUT GET PUTI GETI
```

PUTI

```
3:
2: [[ 0.00 0.00 0.00 0...
1:          { 3.00 4.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

Use the [EDIT] function to modify the position index. The modified position index is then [ENTER]ed:

{ 4 4 } [ENTER]

```
3:
2: [[ 0.00 0.00 0.00 0...
1:          { 4.00 4.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

P4 [ENTER]

```
3: [[ 0.00 0.00 0.00 0...
2:          { 4.00 4.00 }
1:               150.00
→ARRY ARRY→ PUT GET PUTI GETI
```

PUTI

```
3:
2: [[ 0.00 0.00 0.00 0...
1:          { 4.00 5.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

Use the [EDIT] function to modify the position index. The modified position index is then [ENTER]ed:

{ 5 5 } [ENTER]

```
3:
2: [[ 0.00 0.00 0.00 0...
1:          { 5.00 5.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

P5 [ENTER]

```
3: [[ 0.00 0.00 0.00 0...
2:          { 5.00 5.00 }
1:               200.00
→ARRY ARRY→ PUT GET PUTI GETI
```

PUTI

```
3:
2:  [[ 0.00 0.00 0.00 0...
1:          { 1.00 1.00 }
→ARRY ARRY→ PUT GET PUTI GETI
```

Drop the index string.

DROP

```
1:  [[ 0.00 0.00 0.00 0...
     [ 0.00 50.00 0.00 ...
     [ 0.00 0.00 100.00...
→ARRY ARRY→ PUT GET PUTI GETI
```

Store matrix P.

'P STO

```
3:
2:
1:
→ARRY ARRY→ PUT GET PUTI GETI
```

Store the total number of trees sustained in variable S.

120 ENTER

```
3:
2:
1:                  120.00
→ARRY ARRY→ PUT GET PUTI GETI
```

'S STO

```
3:
2:
1:
→ARRY ARRY→ PUT GET PUTI GETI
```

Compute the 5×1 growth ratio matrix GG.

Enter $gg_1 = 0$.

0 ENTER
'GG1 STO

```
3:
2:
1:
→ARRY ARRY→ PUT GET PUTI GETI
```

Compute $gg_2 = 1/g_1$.

USER G1
1/x

```
3:
2:
1:                    2.50
GG1 G4 G3 G2 G1 S
```

'GG2 STO

```
3:
2:
1:
GG2 GG1 G4 G3 G2 G1
```

Compute $gg_3 = 1/g_1 + 1/g_2$

GG2
G2
[1/x]

```
3:
2:                 2.50
1:                 1.54
 GG2 | GG1 | G4 | G3 | G2 | G1
```

[+]

```
3:
2:
1:                 4.04
 GG2 | GG1 | G4 | G3 | G2 | G1
```

'GG3 [STO]

```
3:
2:
1:
 GG3 | GG2 | GG1 | G4 | G3 | G2
```

Compute $gg_4 = 1/g_1 + 1/g_2 + 1/g_3$

GG3
G3
[1/x]

```
3:
2:                 4.04
1:                 3.50
 GG3 | GG2 | GG1 | G4 | G3 | G2
```

[+]

```
3:
2:
1:                 7.54
 GG3 | GG2 | GG1 | G4 | G3 | G2
```

'GG4 [STO]

```
3:
2:
1:
 GG4 | GG3 | GG2 | GG1 | G4 | G3
```

Compute $gg_5 = 1/g_1 + 1/g_2 + 1/g_3 + 1/g_4$

GG4
G4
[1/x]

```
3:
2:                 7.54
1:                 7.50
 GG4 | GG3 | GG2 | GG1 | G4 | G3
```

[+]

```
3:
2:
1:                15.04
 GG4 | GG3 | GG2 | GG1 | G4 | G3
```

'GG5 [STO]

```
3:
2:
1:
 GG5 | GG4 | GG3 | GG2 | GG1 | G4
```

Now invert gg_2, gg_3, gg_4, and gg_5 to form the actual entries into matrix GG.

GG2
1/x

```
3:
2:
1:                 0.40
GG5 GG4 GG3 GG2 GG1 G4
```

'GG2 STO

```
3:
2:
1:
GG5 GG4 GG3 GG2 GG1 G4
```

GG3
1/x

```
3:
2:
1:                 0.25
GG5 GG4 GG3 GG2 GG1 G4
```

'GG3 STO

```
3:
2:
1:
GG5 GG4 GG3 GG2 GG1 G4
```

GG4
1/x

```
3:
2:
1:                 0.13
GG5 GG4 GG3 GG2 GG1 G4
```

'GG4 STO

```
3:
2:
1:
GG5 GG4 GG3 GG2 GG1 G4
```

GG5
1/x

```
3:
2:
1:                 0.07
GG5 GG4 GG3 GG2 GG1 G4
```

'GG5 STO

```
3:
2:
1:
GG5 GG4 GG3 GG2 GG1 G4
```

Create the 5×1 matrix GG. Put the elements on the stack.

GG1
GG2
GG3
GG4
GG5

```
3:                 0.25
2:                 0.13
1:                 0.07
 GG5  GG4  GG3  GG2  GG1  G4
```

Enter the matrix dimensions.

{5 1 [ENTER]

```
3:                 0.13
2:                 0.07
1:        { 5.00 1.00 }
 GG5  GG4  GG3  GG2  GG1  G4
```

Create the matrix.

[ARRAY]
→ARRY

```
1: [[ 0.00 ]
    [ 0.40 ]
    [ 0.25 ]
 →ARRY ARRY→ PUT GET PUTI GETI
```

Store matrix GG.

'GG [STO]

```
3:
2:
1:
 →ARRY ARRY→ PUT GET PUTI GETI
```

Write a program to compute the yield vector.

« S P × GG × » [ENTER]

```
3:
2:
1:        « S P * GG * »
 →ARRY ARRY→ PUT GET PUTI GETI
```

Store program YLD.

'YLD [STO]

```
3:
2:
1:
 →ARRY ARRY→ PUT GET PUTI GETI
```

Compute the 5×1 yield vector YL.

[USER]
YLD

```
1: [[ 0.00 ]
    [ 2400.00 ]
    [ 2971.43 ]
 YLD  GG  GG5  GG4  GG3  GG2
```

You can use [EDIT] or [VIEW↓] to view the entire vector.

$$YL = \begin{bmatrix} 0 \\ 2400.00 \\ 2971.43 \\ 2387.75 \\ 1595.91 \end{bmatrix}$$

The resulting yield vector shows that height class 3 should be harvested to maximize the annual sustainable yield, since yl_3 = \$2971.43 is the maximum entry.

Purge the user variables created in this problem section.

Step-by-Step Examples for Your HP-28C

Vectors and Matrices contains a variety of examples and solutions to show how you can solve your technical problems more easily.

- General Matrix Operations
  Matrix Addition, Multiplication, Determinant, Inverse, Transpose, Conjugate, Minor, Rank, Hermitian Matrices
- Systems of Linear Equations
  Non-homogeneous and Homogeneous Systems, Iterative Refinement
- Vector Spaces
  Basis, Orthogonality, Vector Length, Normalization, Orthogonalization, Orthonormal Basis
- Eigenvalues
  Characteristic Polynomial, Eigenvalues, Eigenvectors
- Least Squares
  Straight Line Fitting, Quadratic Polynomial
- Markov Chains
  Steady State of a System
- An Example: Forest Management Model and Yield

HEWLETT PACKARD

Reorder Number
00028-90044

00028-90059
Printed in U.S.A. 3/87

0 88698 00004 5