Reference Manual

APL★PLUS†/PC Financial and Statistical Library

by Ralph L. Fox
edited by Mary R. Wise

STSC, Inc., A CONTEL Company
2115 East Jefferson Street
Rockville, Maryland 20852
(301) 984-5000

This document was prepared to assist users of STSC's APL*PLUS Service and software products; its contents may not be used for any other purpose without written permission. The material contained herein is supplied without representation or warranty of any kind. STSC, Inc., therefore assumes no responsibility and shall have no liability of any kind arising from the supply or use of this document or the material contained herein.

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America.

CONTENTS

INTRODUCTION 1

1. PROGRAM SUMMARIES 1-1

The *FINANCE* Workspace 1-2
The *FORECAST* Workspace 1-13
The *PROBABILITY* Workspace 1-16
The *STATISTICS* Workspace 1-16

2. THE *FINANCE* WORKSPACE 2-1

ASSETS 2-2
BONDS 2-4
CALCULATE 2-6
CASH 2-18
CHANGETIME 2-22
CONTRATES 2-29
CTIMEVALUE 2-32
DEBT 2-34
DEPRECIATION 2-44
DISTRIBUTE 2-51
INTEREST 2-55
INVENTORY 2-58
MANIPULATE 2-61
MATRIX 2-66
MINMAX 2-67
PTIMEVALUE 2-69
RATES 2-72
RATIOS 2-88
ROUNDING 2-89
TAXES 2-91
TESTS 2-93
TIMEVALUE 2-97

3. THE *FORECAST* WORKSPACE 3-1

CYCLE 3-2
DERIVETREND 3-4
EXPOSMOOTH 3-10
EXPOTREND 3-13
FORETREND 3-15
RELATION 3-18
SMOOTHING 3-20
TREND 3-22

4. THE *PROBABILITY* WORKSPACE 4-1

EMPIRICAL 4-2
ANALYTIC 4-7
DISTFN 4-42

5. THE *STATISTICS* WORKSPACE 5-1

BASICSTATS 5-2
CORRELATE 5-5
REGRESSION 5-10

INDEX OF PROGRAMS AND GROUPS I-1

INTRODUCTION

The APL*PLUS/PC Financial and Statistical Library consists of more than 200 APL routines (programs) that allow you to

- perform financial calculations
- forecast time series
- compute statistical measures
- generate probability distribution data
- manipulate data arrays.

The routines can be used on a stand-alone basis or they can be integrated into other APL applications.

This manual is directed toward financial analysts, management consultants, and application developers. You should have rudimentary APL skills to make full use of the Library (see What You Need To Know About APL).

Organization of The Manual And Documentation Conventions

The Reference Manual comprises this Introduction, five chapters, and an alphabetical index of routines. Chapter 1 contains brief summaries of all the routines in the Financial and Statistical Library. Chapter 2 describes the routines in the *FINANCE* workspace, Chapter 3 describes the routines in the *FORECAST* workspace, Chapter 4 describes the routines in the *PROBABILITY* workspace, and Chapter 5 describes the routines in the *STATISTICS* workspace. The manual states and illustrates the algorithms employed; it is not intended as a tutorial in either the theory or applicability of the Library routines.

The types of information included in the detailed descriptions are

- Program: the name of the APL routine.

- Group: the category that contains several functionally related routines. These groups have been formed to help you search the Library for particular capabilities; they are not groups in the APL sense.

- Syntax: a symbolic representation of how a routine should be used. The syntax depicts whether the routine takes a right or left argument, or returns an explicit, assignable result.

- Subroutines: a complete list of all other routines required to run the routine under consideration. To use a routine in a different operating environment, copy it and all of its subroutines from the Library workspace and save them into the new workspace.

- Description: a brief summary of what the routine does.

- Input: a description of the routine's argument(s) as shown in the syntax.

- Output: a description of the explicit result (if any), any global variables produced, or other information printed (if any) when the routine is run. Discussion of the algorithms employed is usually included in this section.

- Examples: a sequence of actual executions that illustrates most operational variants.

- Notes: any supplementary theoretical information, special pointers or cautions, or amplifications on the examples.

In the examples in the manual, user entries are indented six spaces to distinguish them from system responses. This convention matches system behavior. However, where user entries exceed the page width, the second line is indented three spaces. (There is no indent on the system in this case.) When a system response exceeds the page width, the second line is indented six spaces, matching system behavior.

How To Get Started

The Financial and Statistical Library runs under the APL*PLUS/PC Application Development System, which runs on the IBM Personal Computer and certain other personal computers. Your personal computer should have at least 256K of RAM memory and one disk drive. (If you have less than 256K of RAM memory, you will be able to copy individual routines from the workspaces on the disk, but you will not be able to load an entire workspace.) The Library consists of one disk containing the application software (four APL workspaces on side one of the disk) and this Reference Manual.

To prepare your computer to use the Library, you must load the Disk Operating System (DOS) and the APL*PLUS/PC System. (Refer to the installation instructions that came with your APL*PLUS/PC System.) You do not need to leave the APL*PLUS/PC System disk in a drive unless you plan to use a system HELP facility file. (The HELP facility is only available with Release 3 of the APL*PLUS/PC System.)

Insert the Financial and Statistical Library disk in an available drive. If the disk is in disk drive A, you load workspaces with the APL library number 0 (zero). For example, to load the *FINANCE* workspace, enter:

```
      )LOAD 0 FINANCE
```

If the disk is in disk drive B, use the library number 1, and so on.

To copy particular routines from the workspace, enter:

```
      )COPY 0 FINANCE DIROR DCF IDF IROR
```

You can omit the library number if the disk is in your default disk drive. Once you have loaded one of the workspaces, you can begin to use the routines in that workspace.

Because of the eight-character limit on workspace names in the APL*PLUS/PC System, the *PROBABILITY* and *STATISTICS* workspaces appear in the software as *PROBABIL* and *STATISTI*.

What You Need To Know About APL

The APL knowledge needed for direct use of the Library (as illustrated in this manual) is summarized here.

Data is assigned to variables (without display) with the left arrow. The data in the variable can be displayed just by entering its name. Data vectors are formed by entering actual values separated by spaces or by using the reshape function (⍴) to recycle partial data to the length specified by the left argument.

```
      VECTOR1 ← 2 4 8 16 32
      VECTOR1
2 4 8 16 32

      VECTOR2 ← 12⍴4 4 5
      VECTOR2
4 4 5 4 4 5 4 4 5 4 4 5
```

Matrices are formed by reshaping data to the shape of the left argument. The length or shape of an array can be determined by using ⍴ without a left argument.

```
      MATRIX ← 3 4⍴VECTOR1
      MATRIX
 2  4  8 16
32  2  4  8
16 32  2  4

      ⍴VECTOR1
5

      ⍴VECTOR2
12
```

```
      ρMATRIX
3 4
```

The result of a routine can be assigned to a variable or used in another routine only if the syntax of the routine indicates an explicit result (for example, *R ← LEVPAY X*).

Data vectors are joined to make a longer vector using the catenation function (,).

```
      1,VECTOR1,64 128
1 2 4 8 16 32 64 128

      ρVECTOR1,VECTOR2
17
```

Indexing of arrays is done with square brackets. Semicolons separate the row and column index sets for matrices. Indexed assignment can be used to change particular elements of a vector or matrix.

```
      VECTOR1[4 1 3 4]
16 2 8 16

      VECTOR1[2 4]←22 44
      VECTOR1
2 22 8 44 32

      MATRIX[2 3;2 1]
  2 32
 32 16

      MATRIX[2;]
32 2 4 8

      MATRIX[;4]←41 42 43
      MATRIX
  2  4  8 41
 32  2  4 42
 16 32  2 43
```

The symbols for the common arithmetic functions are +, -, ×, and ÷. The star (*) is used for exponentiation. These functions apply to entire arrays of data. Arithmetic functions performed between a scalar (single number) and an array are performed as though the scalar were extended to the shape of the array. Negative numbers are formed with the high minus sign (¯, SHIFT 2). The middle minus sign (-, SHIFT +) indicates subtraction if data is on the left, or negates the data on the right if there is no left argument.

All APL evaluations (arithmetic or otherwise) are performed from rightmost to leftmost function, but this order can be overridden using parentheses. There is no inherent order of precedence. Any routine, function, parenthesis, or bracket separates an APL expression into its executable components (including commas for catenation, but not spaces in a numeric vector). The following examples illustrate the APL order of execution.

```
      2+5×7               (× then +)
37

      5×7+2               (+ then ×)
45

      (5×7)+2             (× then +)
37

      5×7 8+2             (+ then ×, both with)
45 50                      scalar extensions)

      3 5×2 4             (× on two vectors)
6 20

      3 5×2,4             (, then ×)
6 20

      2,4×3 5             (× then ,)
2 12 20

      (2,4)×3 5           (, then ×)
6 20

      3 5×8 ¯2            (× is the only function)
24 ¯10

      3 5×8 - 2           (- then ×)
18 30

      ⍴VECTOR2,VECTOR1    (, then ⍴)
17

      (⍴VECTOR2),VECTOR1 (⍴ then ,)
12 2 22 8 44 32
```

If you want to write some custom APL programs, you will need to know the basics of program definition mode, local and global variables, conversational input, results formatting, and workspace management. If you will be dealing with large volumes of data, you may want to learn how to use the APL*PLUS File System.

A good introductory tutorial on these topics is APL Is Easy!, one of the books included with the APL*PLUS/PC System.

How To Use The Library

The Financial and Statistical Library is a diverse collection of routines with thousands of variations. Most are non-conversational: you supply data and parameters (interest rates, periodicities, algorithm switches, and so on) to the routines in the form of right and left arguments. These arguments can be actual numeric values, variables assigned appropriate values, or any APL expressions that produce the appropriate values, including other routines from the Library. You can use the routines directly so that the results are calculated and possibly displayed immediately, or you can incorporate them into your own programs.

In the first case, you can have complete control of the sequence of the computations and base your decisions about further calculations on the intermediate results. You may need several related routines to perform an analysis, using the results of some routines as the input to others.

In the second case, you can "package" several routines into a program that you write. In such a package, or "application", a Library routine would not be invoked immediately, but would be executed when your program arrives at the step containing the routine. Other parts of a custom program typically include a section that prompts for data, parameters, and options; and a section that formats the results clearly.

Both of these approaches are illustrated here using the example of a home mortgage analysis. You could use the ad-hoc, immediate execution approach as follows:

- Use the *LEVPAY* program to compute the monthly interest on a $50,000 mortgage loan for 25 years (300 months) at 11% interest and assign the result to the variable *INT*. Compare the example to the documentation for the *LEVPAY* program found on page 2-36. Note that *LEVPAY* not only applies to home mortgage loans, but also to any fixed payment loan of arbitrary duration.

```
      INT ← LEVPAY 1 50000 300 .11
```

- Assign the monthly principal stream and the total monthly payment to the variables *PRIN* and *PAY*.

```
      PRIN ← Z1   (Global result of LEVPAY)
      PAY ← PRIN + INT
```

- Look at the numbers.

```
      INT
458.3333333 458.0425373 . . . 4.451380614
      PRIN
31.72320513 32.01400117 . . . 485.6051578
      PAY
490.0565385 490.0565385 . . . 490.0565385
```

- Determine the totals with the *TOTAL* program.

```
      TOTAL INT
97016.96154

      TOTAL PRIN
50000

      TOTAL PAY
147016.9615
```

- Convert these results to yearly subtotals and print a table showing cumulative results to two decimal places using the *MQY* program and the APL*PLUS System function ⎕*FMT*.

```
      YRINT ← 1 12 MQY INT
      YRPRIN ← 1 12 MQY PRIN
      'CF11.2' ⎕FMT (YRINT;CUM YRINT;PRIN;CUM YRPRIN)
   5,480.21    5,480.21       31.72     400.47
   5,433.87   10,914.08       32.01     847.28
   5,382.16   16,296.24       32.31   1,345.80
   .
   .
   .
   1,426.43   95,770.07       38.78  39,485.53
     910.99   96,681.06       39.13  44,455.22
     335.90   97,016.96       39.49  50,000.00
```

The left argument of ⎕*FMT* is the format specification; here it indicates the use of commas in the thousands position and a field width of 11, including two decimal places. The right argument contains the data vectors for each of the columns in the resulting table. (Other argument styles are possible.) Calculations (*CUM* in this example) can be applied to data within the format expression.

To extend the scope of your analysis, you might decide to write an interactive application to consider the tax deductibility of your interest payments and determine your net cash outlay after the tax effects. Since you must pay some loan origination "points" and other fees, add your up-front expenses to the analysis. You'd also like to improve your table by incorporating column headings and row labels.

The following simple prompting program returns a five-element vector of the relevant loan information. Note the use of the comment symbol (⍝) and the use of the quote (') to print character data (text).

```
     ∇ R←GETLOANINFO
[1]   ⍝   PROMPTS FOR AND RETURNS VECTOR OF 5 LOAN PARAMETERS
[2]    R← 0 0 0 0 0 ⍝    INITIALIZE RESULT
[3]    'PRINCIPAL AMOUNT' ◇ R[1]←⎕
[4]    'NUMBER OF YEARS' ◇ R[2]←⎕
[5]    'PERCENT INTEREST' ◇ R[3]←⎕
[6]    'NUMBER OF POINTS' ◇ R[4]←⎕
[7]    'OTHER START-UP FEES' ◇ R[5]←⎕
     ∇
```

The program does not attempt data validity checking or error handling. The *INPUT* workspace distributed by STSC contains a recommended set of prompting utilities that could be used for these purposes.

The program *ONELOAN* incorporates the prompting program with several Library routines to complete the application. Note the localized variables in the program header.

```
    ∇ ONELOAN;INFO;INT;OUT;PAY;PRIN;SAVE;YRINT;YRPRIN;YRS;Z1
[1]  ⍝   PERFORMS YEARLY ANALYSIS FOR MONTHLY FIXED RATE MORTGAGE;
[2]  ⍝   ASSUMES REGULAR PAYMENTS START IN JANUARY, WHILE POINTS AND OTHER
[3]  ⍝   START-UP FEES ARE PAID IN DECEMBER OF THE PREVIOUS YEAR.
[4]  ⍝   REQUIRES GLOBAL VARIABLES <STARTYEAR> (YEAR OF THE FIRST JANUARY
[5]  ⍝   PAYMENT) AND <TAXBRACKET> (MARGINAL INCOME TAX PERCENT).
[6]  ⍝   REQUIRES ROUTINES <LEVPAY>, <MQY>, <CUM>, <ROUND>, <INCREASE>,
[7]  ⍝   <PCTOF>, <TIMES>, AND <TO> FROM THE FINANCE WORKSPACE.
[8]  ⍝   ALSO REQUIRES THE PROMPTING ROUTINE <GETLOANINFO>.
[9]  ⍝
[10] ⍝   PROMPT FOR PRINCIPAL AMT, NO. YEARS, INTEREST PCT, POINTS, FEES
[11]  INFO←GETLOANINFO
[12] ⍝   SET UP THE YEARLY LABELS, INCLUDING START-UP YEAR
[13]  YRS←(STARTYEAR-1)+0 TO INFO[2]
[14] ⍝   COMPUTE THE MONTHLY INTEREST, PRINCIPAL, TOTAL PAYMENT
[15]  INT←LEVPAY 1,INFO[1],(12×INFO[2]),INFO[3]÷100
[16] ⍝   LEVPAY PRODUCES GLOBAL VARIABLE Z1
[17]  PRIN←Z1 ⋄ PAY←INT+PRIN
[18]  'MONTHLY PAYMENT IS' ⋄ 0.01 ROUND PAY[1] ⋄ ''
[19] ⍝   CONVERT TO YEARLY TOTALS FOR DISPLAY PURPOSES
[20] ⍝   FIRST INTEREST, INCLUDING 'POINTS' AT SETUP
[21]  YRINT←(INFO[4] PCTOF INFO[1]), 1 12 MQY INT
[22] ⍝   THEN PRINCIPAL., WITH NONE PAID AT SETUP
[23]  YRPRIN←0, 1 12 MQY PRIN
[24] ⍝   FINALLY COMPUTE THE TAX SAVINGS AND NET CASH OUTLAY
[25]  SAVE←YRINT×TAXBRACKET÷100 ⋄ OUT←YRPRIN+YRINT-SAVE
[26] ⍝   ADD THE FEES IN THE FIRST POSITION
[27]  OUT←(1,INFO[5]) INCREASE OUT
[28] ⍝   DISPLAY RESULTS WITH HEADINGS, COMMAS, AND 2 DECIMAL PLACES
[29]  'YEAR   INTEREST    CUM INT  PRINCIPAL   CUM PRIN TAX SAVNGS NET OUTLAY'
[30]  '----   ---------   -------  ---------   -------- ---------- ----------'
[31]  'I4,6CF11.2' ⎕FMT(YRS;YRINT;CUM YRINT;YRPRIN;CUM YRPRIN;SAVE;OUT)
    ∇
```

Before executing the program for the first time, you assign the necessary global variables. Since the starting year and your tax bracket will probably be constant throughout the analysis of several mortgage loans, they are excluded from *GETLOANINFO* and designed into the application as global variables. This means that these variables do not appear in the program header but must be assigned outside of the program. Thus, properly documented global variables are one way of parameterizing options in an application -- making them changeable but not requiring them to be entered each time the application is run.

```
STARTYEAR ← 1983
TAXBRACKET ← 39
```

Now you can run the program.

```
      ONELOAN
PRINCIPAL AMOUNT
⎕:
      50000
NUMBER OF YEARS
⎕:
      25
PERCENT INTEREST
⎕:
      11
NUMBER OF POINTS
⎕:
      3
OTHER START-UP FEES
⎕:
      800
MONTHLY PAYMENT IS
490.06
```

YEAR	INTEREST	CUM INT	PRINCIPAL	CUM PRIN	TAX SAVNGS	NET OUTLAY
----	--------	-------	---------	--------	----------	----------
1982	1,500.00	1,500.00	0.00	0.00	585.00	1,715.00
1983	5,480.21	6,980.21	400.47	400.47	2,137.28	3,743.40
1984	5,433.87	12,414.08	446.81	847.28	2,119.21	3,751.47
1985	5,382.16	17,796.24	498.52	1,345.80	2,099.04	3,781.64
1986	5,324.47	23,120.71	556.20	1,902.00	2,076.55	3,804.13
1987	5,260.11	28,380.82	620.57	2,522.57	2,051.44	3,829.24
1988	5,188.30	33,569.12	692.38	3,214.95	2,023.44	3,857.24
1989	5,108.18	38,677.30	772.50	3,987.45	1,992.19	3,888.49
1990	5,018.79	43,696.09	861.89	4,849.34	1,957.33	3,923.35
1991	4,919.05	48,615.14	961.63	5,810.97	1,918.43	3,962.25
1992	4,807.77	53,422.91	1,072.91	6,883.88	1,875.03	4,005.65
1993	4,683.61	58,106.52	1,197.06	8,080.94	1,826.61	4,054.07
1994	4,545.09	62,651.61	1,335.59	9,416.53	1,772.59	4,108.09
1995	4,390.54	67,042.15	1,490.14	10,906.67	1,712.31	4,168.37
1996	4,218.10	71,260.26	1,662.58	12,569.24	1,645.06	4,235.62
1997	4,025.71	75,285.97	1,854.97	14,424.21	1,570.03	4,310.65
1998	3,811.06	79,097.02	2,069.62	16,493.83	1,486.31	4,394.37
1999	3,571.56	82,668.58	2,309.12	18,802.95	1,392.91	4,487.77
2000	3,304.35	85,972.94	2,576.33	21,379.28	1,288.70	4,591.98
2001	3,006.22	88,979.16	2,874.45	24,253.73	1,172.43	4,708.25
2002	2,673.60	91,652.76	3,207.08	27,460.81	1,042.70	4,937.98
2003	2,302.48	93,955.23	3,578.20	31,039.02	897.97	4,982.71
2004	1,888.41	95,843.64	3,992.27	35,031.28	736.48	5,144.20
2005	1,426.43	97,270.07	4,454.25	39,485.53	556.31	5,324.37
2006	910.99	98,181.06	4,969.69	44,455.22	355.29	5,525.39
2007	335.90	98,516.96	5,544.78	50,000.00	131.00	5,749.68

You might now decide to write another program called *TWOLOANS* that will compare two different mortgage loans. By introducing the inflation rate into the analysis, you could select between the loans based on net present value. You might even use a breakeven routine to determine the number of months necessary for the better loan to establish its superiority. (Such a demonstration program does indeed exist! For full details, enter *∇TWOLOANS[⎕]∇* in the *FINANCE* workspace. The programs *GETLOANINFO* and *ONELOAN* are also in the *FINANCE* workspace.)

Your Comments Are Welcome

We're eager to hear your comments on the Financial and Statistical Library. We've provided a Feedback Form and postage-paid mailer at the back of this manual for your convenience. If the form is missing, send your comments to:

STSC, Inc.
Marketing Communications
2115 East Jefferson Street
Rockville, Maryland 20852

Please include your name, address, and the name of the application software with your comments.

CHAPTER 1

PROGRAM SUMMARIES

The *FINANCE* Workspace 1-2
The *FORECAST* Workspace 1-13
The *PROBABILITY* Workspace 1-16
The *STATISTICS* Workspace 1-20

The *FINANCE* Workspace

Group: *ASSETS*

CASHMGT

Calculates managed cash, investment, and loan positions using specified maximum and minimum balances.

PLANT

Calculates plant depreciation using future plant requirements.

Group: *BONDS*

BOND

Computes a bond amortization schedule using purchase price and coupon parameters. Yields coupon payments, interest earned, and ex-coupon book value.

BPRICE

Computes a bond purchase price (present value) using yield rate and coupon parameters.

BYIELD

Computes the period yield rate for a bond using initial price and coupon parameters.

Group: *CALCULATE*

ACOMPARE

Calculates actual comparative differences between two time series.

PCOMPARE

Calculates percentage comparative differences between two time series.

AFDIFF

Calculates actual first differences between successive time periods of time series data.

PFDIFF

Calculates percentage first differences between successive time periods of time series data.

AVEBAL

Calculates the average balance over successive periods of time series balances.

AVGRED

Reduces a matrix of time series into a single average time series.

CUM

Calculates the cumulative sums of a time series or matrix of time series.

CUMPROD
Calculates the cumulative products of a time series or matrix of time series.

DIV
Extends the division function to handle division by zero and division between vectors and matrices.

DIVRED
Reduces a two-row matrix of time series into its quotient, handling division by zero.

GREATEST
Calculates the largest value in a time series, or the largest values in a matrix of time series.

LEAST
Calculates the smallest value in a time series, or the smallest values in a matrix of time series.

MPROD
Multiplies a time series by several factors, yielding a matrix of all possible combinations of the multiplications.

TIMES
Extends the multiplication function to allow multiplication of a list of factors by a matrix of time series.

PCTOF
Multiplies percentages and an array of data values.

PLUS
Extends the addition function to allow addition between vectors and matrices.

MINUS
Extends the subtraction function to allow subtraction between vectors and matrices.

TOTAL
Totals the values in a time series or a matrix of time series.

AGGREGATE
Totals (down) a matrix of several time series.

WTDSUM
Calculates the weighted sum of a time series or matrix of time series. Parameters are positive or negative weights.

YTD
Calculates cumulative year-to-date sums across multiple years of a time series. Can also be used with matrix data.

Group: *CASH*

BALFOR
Calculates balance forward for balance sheet statistics.

SBALFOR
Calculates balance forward for balance sheet statistics, preserving initial balance in the result.

BREAKEVEN
Calculates the breakeven period given an opening balance and cash flow.

CURRENT
Calculates the current and long term portions of debt using the debt principal payment stream.

PAYMT
Calculates lump-sum payments using a payment stream and parameters of periodicity and payment type.

ΔBAL
Calculates changes in balance accounts.

ΔBALS
Calculates changes in balance accounts with a specified initial change.

Group: *CHANGETIME*

MQY
Converts one or more time series to different periodicity using parameters specifying summing, averaging, or linear interpolation.

MQYFACTOR
Replicates data to reflect a change in periodicity.

FIRST
Sets the starting period positions for data in a time span.

LAST
Set the ending period positions for data in a time span.

HALFYR
Performs a half-year shift on time series data.

MTW
Converts monthly time series data to weekly time series data.

WTM
Converts weekly time series data to monthly time series data.

PRIOR
Produces the values of a time series from a prior period.

SHIFT
Shifts data along a time axis.

Group: *CONTRATES*

CDCF
Calculates discounted cash flow with continuous compounding, using parameters of payment type, periodicity, and discount rate.

CIROR
Calculates internal rate of return using continuous compounding. Parameters allow selection of periodicity and choice of effective annual rate or nominal annual rate.

CIDF
Calculates an internal discount factor using continuous compounding.

Group: *CTIMEVALUE*

CFV
Calculates future value of a cash stream where discount rates are compounded continuously. Parameters are payment type, periodicity, and nominal annual discount rate.

CPV
Calculates present value of a cash stream where discount rates are compounded continuously. Parameters are payment type, periodicity, and nominal annual disount rate.

Group: *DEBT*

EPRIN
Calculates equal principal payment loan data.

GLOAN
Calculates loan data given a payment schedule.

LEVPAY
Calculates level payment loan data.

DLEVPAY
Solves for any missing parameter needed for the *LEVPAY* program.

LEVPAY1

Calculates level payment loan data when the interest rate varies from year to year, or when the principal portion of the payment must be a multiple of a given value.

LOAN

Calculates selectable loan data.

LOFC

Calculates line of credit for a given cash stream. Computes principal and interest, loan balances, fees, and acquisition costs.

TLOAN

Calculates term loan data.

VLOAN

Calculates variable payment schedule loan data.

Group: *DEPRECIATION*

ACRS

Calculates depreciation using the Accelerated Cost Recovery System (ACRS).

DEPRE

Calculates selectable depreciation methods.

STL

Calculates straight line depreciation.

SYD

Calculates sum-of-years digits depreciation.

DBAL

Calculates declining balance depreciation with or without an optimal switch.

SFUND

Calculates the sinking fund method of depreciation.

ACCSF

Calculates the accelerated sinking fund method of depreciation.

Group: *DISTRIBUTE*

LEADLAG

Leads or lags a time series using a time index and multiplicative factors. The program *DAYLAG* can generate the time index and multiplicative factors.

DAYLAG

Produces multiplicative factors for a specified number of days of lead or lag.

QFACTOR

Applies quarterly factors to a monthly time series.

YFACTOR

Applies yearly factors to a monthly time series.

SPREAD

Spreads one time series across another time series.

Group: *INTEREST*

EYIR

Calculates effective yearly interest rates, given nominal rates and number of compoundings per year.

NYIR

Calculates nominal yearly interest rates, given effective rates and number of compoundings per year.

QTR2NDINTEREST

Calculates secondary interest effects of a cash flow using quarterly data.

YR2NDINTEREST

Calculates secondary interest effects of a cash flow using yearly data.

Group: *INVENTORY*

AVGCOST

Calculates cost of goods sold on an average cost basis.

FIFO

Calculates cost of goods sold on a "first-in, first-out" basis.

LIFO

Calculates cost of goods sold on a "last-in, first-out" basis.

DISCOUNT

Calculates discount amounts based on a volume discount schedule.

VOLCOST

Calculates total costs based on a volume discount schedule.

Group: *MANIPULATE*

CORI

Determines whether the resulting values of a time series are calculated or input, and performs the required calculations.

ELTOF

Yields specified elements of a time series.

INCREASE

Allows increments of specific elements of time series.

REPLACE

Allows replacement of specific elements of a time series.

INIT

Replaces initial value(s) of a time series with input value(s).

LZF

Completes a time series to specified number of periods with leading zeros.

RZF

Completes a time series to specified number of periods with trailing zeros.

RLF

Completes a time series to specified number of periods by filling to the right with the last value in the series.

LFF

Completes a time series to specified number of periods by filling to the left with the first value in the series.

TRANSPOSE

Reverses the rows and columns of a matrix of data.

PLUSMINUS

Selects only positive or only negative numbers from a series, or converts numbers to absolute values.

Group: *MATRIX*

BCMATX

Builds a matrix column-by-column.

BRMATX

Builds a matrix row-by-row.

Group: *MINMAX*

MIN

Compares two time series and yields a time series containing the minimum values from each period.

MAX

Compares two time series and yields a time series containing the maximum values from each period.

MINSCAN

Compares successive periods of a time series and replaces the higher value of each comparison with the lower value.

MAXSCAN

Compares successive periods of a time series and replaces the lower value of each comparison with the higher value.

Group: *PTIMEVALUE*

PDCF

Calculates expected value of cash flow, compounded every period, with parameters of periodicity, payment type, nominal discount rates, and probabilities for each flow.

PFV

Calculates the probabilistic future value of a matrix of cash streams using interest rates and probabilities.

PPV

Calculates the probabilistic present value of a matrix of cash streams using interest rates and probabilities.

Group: *RATES*

DCF

Calculates generalized discounted cash flow with discounting compounded each period.

FV

Calculates the future value of a cash stream using nominal annual interest rates.

PV

Calculates the present value of a cash stream using nominal annual interest rates.

ROR

Calculates rate of return; allows selection of type of rate of return under program control.

IROR

Calculates internal rate of return; used by other rate of return programs.

MIROR

Calculates internal rates of return for a matrix of data.

AROR

Calculates annuity rate of return based upon the present value of all equity investments and negative flows discounted at specified rates.

DIROR

Calculates discounted internal rate of return on a cash flow discounted at specified rates.

MSAROR

Calculates a modified savings account rate of return, applying positive benefits to negative flows prior to reinvestment.

SAROR

Calculates savings account rate of return compounding positive benefits forward at specified reinvestment rates.

SFROR

Calculates sinking fund rate of return, applying reinvested positive benefits to future negative flows.

DCIROR

Calculates internal rate of return for a combination of discrete and continuous cash flows.

IDF

Calculates the internal discount factor of a cash stream, assuming no reinvestment of positive benefits.

MIDF

Calculates the internal discount factor of a cash stream matrix of data, assuming no reinvestment of positive benefits.

DCIDF

Calculates an internal discount factor for a combination of discrete and continuous cash flows.

SFCFL

Calculates a sinking fund cash flow, setting aside positive flows to fund future negative flow. Parameters are periodicity and reinvestment rate.

Group: *RATIOS*

INCOMEASSETS
Calculates the ratio of Income Statement items to Balance Sheet items.

PER
Calculates percentages between one time series and another.

Group: *ROUNDING*

RND
Rounds a time series to a specified number of decimal places.

ROUND
Rounds a time series or matrix of time series with respect to a multiple of a specified value.

ROUNDUP
Converts an amount to the nearest specified value above the original quantities.

ROUNDDOWN
Converts an amount to the nearest specified value below the original quantities.

Group: *TAXES*

FEDTAX79
Computes Federal corporate taxes, based upon the 1979 tax law. Includes optional tax loss carry forward.

ITC
Calculates investment tax credit and recaptures if stated life differs from actual life.

Group: *TESTS*

CUMRANGE
Calculates a series of cash balances from time series cash flow numbers keeping the resulting balance within specified maximums and minimums.

CUMZEROMAX
Calculates cash balances from time series cash flow, keeping the minimum balance above zero.

INTRANGE
Tests a time series for values that are whole numbers and within a specified range. Returns a result of 0 or 1 for each number tested.

TRANGE

Tests a time series to determine if numbers are within a specified range.

RANGE

Converts a time series to keep all numbers within a specified range.

MAXHALT

Suspends a model execution when upper bound is exceeded.

MINHALT

Suspends a model execution when lower bound is exceeded.

Group: *TIMEVALUE*

ANNUITY

Calculates various uniform annuities: present value, periodic values, or ultimate value. Parameters are periodicity, payment, rates, and payment-type switch.

Group: *CYCLE*

DSEASFACTOR
Derives seasonal adjustment factors for a time series.

SEASFACTOR
Seasonalizes a time series using seasonal or cyclical factors.

Group: *DERIVETREND*

DTTREND1
Derives first order (linear) time trend coefficients.

DTTREND2
Derives second order (quadratic) time trend coefficients.

DSTEPTREND
Derives starting value, slope, and number of elements as parameters for the *STEPTREND* program.

DPOWER
Derives polynomial (power series) time trend coefficients.

DGROWTH
Derives sets of growth rate parameters using least squares fit(s).

AGR
Derives annual growth rate parameters for monthly, quarterly, or yearly time series.

Group: *EXPOSMOOTH*

EXPOSMOOTH1
Performs single (constant model) exponential smoothing.

EXPOSMOOTH2
Performs double (linear model) exponential smoothing.

EXPOSMOOTH3
Performs triple (quadratic model) exponential smoothing.

Group: *EXPOTREND*

EXPOGROWTH
Forecasts based upon exponential growth. Parameters are number of terms, initial value, and coefficient of time.

ASYMPTOTE

Forecasts asymptotically. Parameters are number of periods, coefficient of time, and upper limit.

SCURVE

Generates values along a saturation (logistic) curve. Parameters are initial and intermediate time periods and values, ending time period, and either the maximum value or slope.

Group: *FORETREND*

FTTREND1

Forecasts based upon a first order (linear) time trend.

FTTREND2

Forecasts based upon a second order (quadratic) time trend.

FPOWER

Forecasts based upon a polynomial (power series) time trend.

FGROWTH

Forecasts based upon a compound growth model using a least squares fit.

Group: *RELATION*

FDIFF

Forecasts based on the average historical difference of two time series.

FREGRESS

Forecasts based on a regression of two time series, one independent and one dependent.

FRATIO

Forecasts based on the average historical ratio of two time series.

FTRENDRATIO

Forecasts based upon the first order (linear) time trend of the historical ratio of two time series.

Group: *SMOOTHING*

MOVINGAVE

Calculates the moving average of a time series, using specified number of periods and either midpoint, leading edge, or trailing edge periods.

MOVMAXSCAN

Calculates moving maximum with variable effect width.

MOVMINSCAN
Calculates moving minimum with variable effect width.

WEIGHTDAVE
Calculates the weighted moving average of a time series, using number or periods, weights, and the same averaging options as the program *MOVINGAVE*.

Group: *TREND*

TTREND1
Forecasts first order time trend using slope and intercept.

TTREND2
Forecasts second order time trend using slope and intercept.

STEPTREND
Forecasts using a step trend function. Parameters are height, slope, and number of terms. Parameters can be derived using the program *DSTEPTREND*.

POWER
Forecasts using an nth order (power series) time trend.

GROWTH
Forecasts using various growth rates, using base value and monthly, quarterly, half-yearly, or yearly growth rates.

DORG
Forecasts by amount, growth rate, or a combination of the two methods.

Group: *EMPIRICAL*

CUMDISCRETE

Returns samples from the discrete distribution; allows specification of cumulative probability parameters and number of samples.

DISCRETE

Returns samples from the discrete distribution; allows specification of probability parameters and number of samples.

HISTOGRAM

Returns random samples from a histogram distribution. Parameters are endpoints and heights.

POLYGON

Returns samples from a polygon distribution. Parameters are endpoints and probabilities, or endpoints and heights.

POLYGONCUM

Returns samples from a histogram distribution whose cumulative distribution function is a polygon. Parameters are endpoints and probabilities, or endpoints and heights.

CUMHISTOGRAM

Returns samples from a histogram distribution whose cumulative distribution function is a polygon. Parameters are endpoints and probabilities, or endpoints and heights.

Group: *ANALYTIC*

PBETA

Returns samples from the Beta distribution using power parameters.

PBINOMIAL

Returns samples from the binomial distribution using parameters of probability of success of each trial.

PCAUCHY

Returns samples from the Cauchy distribution using parameters of mean and width.

NCAUCHY

Returns a specified number of samples from the standard Cauchy distribution.

PCHISQUARED

Returns samples from the central chi-square distribution using parameters of degrees of freedom.

PNONCENCHISQ
Returns samples from the non-central chi-square distribution.

PEXPONENTIAL
Returns samples from the exponential distribution using parameters of mean(s).

NEXPONENTIAL
Returns a specified number of samples from the standard exponential distribution.

NEXTREME
Returns specified number of samples from the standard extreme distribution.

PF
Returns samples from the F-distribution using parameters of degrees of freedom.

PNON and *PCENTRALF*
Returns samples from the non-central F-distribution using the non-centrality parameter and the two degrees of freedom.

PGAMMA
Returns samples from the Gamma distribution using power parameters.

PGEOMETRIC
Returns samples from the geometric distribution using the probability of success of each trial.

PGEOMETRIC1
Returns samples from the geometric distribution using a ratio based on the probability of success of each trial.

PHARMONIC
Returns samples from the harmonic distribution.

PHYPER and *PGEO*
Returns samples from the hypergeometric distribution using parameters of number of samples, number of special items, and population.

PLAPLACE
Returns samples from the Laplace distribution using parameters of mean and width.

NLAPLACE
Returns a specified number of samples from the standard Laplace distribution.

PLDECR
Returns samples from the linear decreasing distribution.

PLINCR
Returns samples from the linear increasing distribution.

PLOGISTIC
Returns samples from the logistic distribution using parameters of mean and width.

NLOGISTIC
Returns a specified number of samples from the standard logistic distribution.

PNORMAL
Returns samples from the normal distribution using parameters of mean and standard deviation.

NNORMAL
Returns a specified number of samples from the standard normal distribution.

PLOGNORMAL
Returns samples from the lognormal distribution using parameters of mean and standard deviation.

PLOGNORMAL1
Returns samples from the lognormal distribution using parameters of mean and standard deviation of the associated normal distribution(s).

PPASCAL
Returns samples from the Pascal distribution using parameters of number of successes and probability of success.

PNEGBIN
Returns samples from the negative binomial distribution using parameters of number of successes and a ratio based on the probability of success.

PORDER
Returns samples from the order distribution using parameters of sample size and number of extremal observations.

PPOISSON
Returns samples from the Poisson distribution using parameters of means.

PSWITCH
Returns samples from the switch distribution using parameters of probabilities.

PT
Returns samples from the t-distribution using parameters of degrees of freedom.

PNONCENTRALT
Returns samples from the non-central t-distribution using parameters of means and degrees of freedom.

PTRIANGULAR

Returns samples from the triangular distribution using parameters of endpoint values.

PUNIFORM

Returns samples from the uniform distribution using lower and upper limits.

NUNIFORM

Returns a specified number of samples from the standard uniform distribution.

PWEIBULL

Returns samples from the Weibull distribution using power parameters.

Group: *DISTFN*

TDIST

Computes the Student's t-distribution using a vector of scores and degrees of freedom.

Group: *BASICSTATS*

DSTAT

Displays basic statistics for a series of numbers, including sample size, maximum, minimum, range, mean, variance, standard deviation, mean deviation, median, and mode.

FREQ

Calculates frequencies for a discrete empirical distribution.

FREQDIST

Calculates frequencies for continuous empirical data.

PCTILES

Calculates percentiles for a series of numbers.

Group: *CORRELATE*

ACORR

Calculates estimates of the autocorrelation function of a series of numbers.

ACOVAR

Calculates an estimate of the autocovariance function of a series of numbers.

CM

Calculates a matrix of correlation coefficients.

CORR

Calculates simple and partial correlation coefficients based on the result of the program *CM*.

PCORR

Calculates simple and partial correlation coefficients based on a matrix of sample data.

CCORR

Calculates cross correlations where one data vector shifts in period with respect to another.

Group: *REGRESSION*

REG

Performs single and multiple regression. Yields table showing variable number, regression coefficient, standard error of the regression coefficient, t-value, beta coefficient, degrees of freedom, sum of squares, mean square, and F-statistic.

DSTATREG

Prints a report of the results of multiple regression.

STEPREG

Generates user-controlled stepwise regression.

STREG

Calculates simple stepwise regression for a matrix of data.

REGRESS

Performs simple regression only.

RES

Calculates table of residuals from a regression.

STATRES

Calculates statistics on the residuals of a regression.

DSTATRES

Generates a descriptive display of statistics on the residuals from a regression, including sum, sum of absolutes, sum of squares, Durbin-Watson statistic, number of runs (+ and -), expected number of runs, and standard deviation.

CHAPTER 2

THE *FINANCE* WORKSPACE

The *FINANCE* workspace contains programs that perform calculations relating to loan amortization, depreciation, internal rate of return, inventory costs, and so on. It also contains a more general class of routines that help calculate and manipulate data arrays. A number of programs are "English" cover functions for APL functions and operators, but their consistent use will make your custom programs easier to write and maintain.

ASSETS 2-2
BONDS 2-4
CALCULATE 2-6
CASH 2-18
CHANGETIME 2-22
CONTRATES 2-29
CTIMEVALUE 2-32
DEBT 2-34
DEPRECIATION 2-44
DISTRIBUTE 2-51
INTEREST 2-55
INVENTORY 2-58
MANIPULATE 2-61
MATRIX 2-66
MINMAX 2-67
PTIMEVALUE 2-69
RATES 2-72
RATIOS 2-88
ROUNDING 2-89
TAXES 2-91
TESTS 2-93
TIMEVALUE 2-97

Program: *CASHMGT* Group: *ASSETS*

Syntax: *R*←Y *CASHMGT X* Subroutines: None

Description: Calculates managed cash and funds positions.

Input: *X* = Cash balances produced by operations

Y[1] = Maximum desired balance for cash on hand

Y[2] = Minimum desired balance for cash on hand

Y[3] = Opening short term investments (marketable securities)

Y[4] = Opening short term debt

Output: *R* = Managed cash balances
Z1 = Managed short term investments (global variable)
Z2 = Managed short term loans (global variable)

If funds are needed to meet the minimum cash requirement, short term investments are liquidated and short term debt is assumed, if necessary.

If an excess of funds over the desired maximum exists, the short term loans are paid off. If there is still an excess, short term investments are made.

Example:

```
      200 100 50 30 CASHMGT ¯100 0 100 200 300
 100 100 120 200 200
      Z1
0   0   0  20 120
      Z2
 180  80   0   0   0
```

Program: *PLANT* Group: *ASSETS*

Syntax: *R*←*Y* *PLANT* *X* Subroutines: None

Description: Calculates depreciation and investments for plant requirements.

Input: If the model span equals *N* periods, then:

X[1 *TO* *N*] = Depreciation schedule for assets held at the opening of the model span.

X[*N*+1 *TO* 2*N*] = Depreciation fraction to be applied against assets "purchased" during the model span. Depreciation is applied in the period following the purchase period.

X[2*N*+1 *TO* 3*N*] = Gross (*Y*[3]=0) or net (*Y*[3]=1) plant requirements over the model span.

Y[1] = Opening gross assets

Y[2] = Opening accumulated depreciation

Y[3] = Calculation type:
0 ←→ gross plant requirements input in *X*
1 ←→ net plant requirements input in *X*

Output: *R* = Depreciation for the *N* periods
*Z*1 = "Investments" made by the model for the *N* periods (global variable)

Examples: In these examples, *N* = 3.

```
      200 50 0 PLANT 40 35 30 .25 .2 .15 260 300
   320
40 50 52
      Z1
60 40 20

      200 50 1 PLANT 40 35 30 .25 .2 .15 260 300
   320
40 72.5 88.125
      Z1
150 112.5 108.125
```

Program: *BOND* Group: *BONDS*

Syntax: *R←BOND X* Subroutine: *IDF*

Description: Computes a bond amortization schedule.

Input: *X*[1] = Bond purchase price

X[2] = Periodic coupon payment

X[3] = Number of coupons remaining

X[4] = Optional -- Redemption value of the bond (default = 1000)

X[5] = Optional -- Fractional part of the first coupon due to the buyer (default = 1)

Output: *R* = Coupon payments paid
Z1 = Interest earned by the bond (global variable)
Z2 = Book value of the bond ex-coupon (global variable)

Examples:

```
      BOND 981.19 20 4
20 20 20 20
      Z1
24.643 24.759 24.878 25.000
      Z2
985.720 990.363 995.122 1000.000

      BOND 981.19 20 4 1050
20 20 20 20
      Z1
36.879 37.503 38.150 38.821
      Z2
997.467 1014.346 1031.850 1050.000

      BOND 981.19 20 4 1050 .6
12 20 20 20
      Z1
23.145 39.297 40.047 40.827
      Z2
992.081 1010.656 1029.953 1050.000
```

Program: *BPRICE* Group: *BONDS*

Syntax: *R←BPRICE X* Subroutines: None

Description: Computes a bond purchase price.

Input: *X*[1] = Period yield rate
X[2] = Periodic coupon amount
X[3] = Number of coupons remaining
X[4] = Optional -- Redemption value of the bond (default = 1000)
X[5] = Optional -- Fractional part of the first coupon due to the buyer (default = 1)

Output: *R* = Present value of the bond

Examples:

```
      BPRICE .025 20 4
981.190

      BPRICE .025 20 4 1050
1026.488

      BPRICE .025 20 4 1050 .6
1028.719
```

Program: *BYIELD* Group: *BONDS*

Syntax: *R←BYIELD X* Subroutine: *IDF*

Description: Computes the period yield rate for a bond.

Input: *X*[1] = Initial price of the bond
X[2] = Periodic coupon amount
X[3] = Number of coupons remaining
X[4] = Optional -- Redemption value of the bond (default = 1000)
X[5] = Optional -- Fractional part of the first coupon due to the buyer (default = 1)

Output: *R* = Bond yield rate

Examples:

```
      BYIELD 985 50 5
.0535

      BYIELD 985 50 5 1050
.0624

      BYIELD 985 50 5 1050 .6
.0635
```

Program: *ACOMPARE* Group: *CALCULATE*

Syntax: *R←Y ACOMPARE X* Subroutines: None

Description: Calculates actual comparative differences.

Input: *X* = Time series
Y = Time series of the same length as *X*

Output: *R* = *Y*-*X*

Example:
```
      10 10 10 10 ACOMPARE 5 6 7 8
5 4 3 2
```

Program: *PCOMPARE* Group: *CALCULATE*

Syntax: *R←Y PCOMPARE X* Subroutine: *DIV*

Description: Calculates percent comparative differences.

Input: *X* = Time series
Y = Time series of the same length as *X*

Output: *R* = (*Y*-*X*)÷*X*

Example:
```
      10 10 10 10 PCOMPARE 5 6 7 8
1 .667 .429 .25
```

Program: *AFDIFF* Group: *CALCULATE*

Syntax: *R←AFDIFF X* Subroutines: None

Description: Calculates actual first differences.

Input: *X* = A single vector time series or a matrix whose rows are several time series

Output: *R* = Actual differences between successive time periods

Examples:

```
          AFDIFF 10 13 15 16 16
    3 2 1 0

          X
     7 38 23 27 11
     3 34 34 47 20
    26 42  2  3 27

          AFDIFF X
    31 ¯15   4 ¯16
    31   0  13 ¯27
    16 ¯40   1  24
```

Program: *PFDIFF* Group: *CALCULATE*

Syntax: *R←PFDIFF X* Subroutine: *DIV*

Description: Calculates percent first differences.

Input: *X* = Time series

Output: *R* = Percent differences `(X[I+1]-X[I])÷X[I])`

Example:

```
          PFDIFF 10 12 15 18
    .2 .25 .2
```

Program: *AVEBAL* Group: *CALCULATE*

Syntax: *R←Y AVEBAL X* Subroutines: None

Description: Calculates the average over successive periods of a stream of balance sheet items.

Input: *X* = Vector stream or matrix whose rows are several streams

Y = Opening balance(s). If *X* is a vector, *Y* is a scalar. If *X* is a matrix, *Y* is a vector with one element for each row of *X*. *Y* is padded with zeros if it is too short.

Output: *R* = Successive averages, of the same shape as *X*

Examples:

```
      80 AVEBAL 100 120 150 200
90 110 135 175

      X
100 120 150 200
140 110 100 120

      80 AVEBAL X
 90 110 135 175
 70 125 105 110

      80 90 AVEBAL X
 90 110 135 175
115 125 105 110
```

Program: *AVGRED* Group: *CALCULATE*

Syntax: *R←AVGRED X* Subroutines: None

Description: "Average reduces" a matrix of time series to a single time series.

Input: *X* = Matrix of time series

Output: *R* = Single time series average of *X*

Example:

```
      X
100 120 150 140 180
150 170 110 210 140
110 190 130 100 130

      AVGRED X
120 160 130 150 150
```

Program: *CUM* Group: *CALCULATE*

Syntax: *R←CUM X* Subroutines: None

Description: Calculates the cumulative sums of a time series or a matrix of time series.

Input: *X*= Vector or matrix

Output: *R* = Cumulative sums of *X* (by rows if *X* is a matrix)

Examples:

```
      CUM 23 16 37 19
23 39 76 95

      X
23 16 37 19
45 43 32 21
18 27 18 31

      CUM X
23  39  76  95
45  88 120 141
18  45  63  94
```

Program: *CUMPROD* Group: *CALCULATE*

Syntax: *R←CUMPROD X* Subroutines: None

Description: Forms a cumulative product.

Input: *X* = Time series

Output: *R* = Cumulative products of *X* (by rows if *X* is a matrix)

Example:

```
      CUMPROD 10 20 30
10 200 6000
```

Program: *DIV* Group: *CALCULATE*

Syntax: *R*←*Y DIV X* Subroutines: None

Description: Extends the division primitive function to handle division by zero and division between vectors and matrices.

Input: *X* = Scalar, time series, or matrix
Y = Scalar, time series, or matrix

If neither *X* nor *Y* is a scalar, lengths must conform.

Output: *R* = *Y*÷*X* when *X*≠0
0 when *X*=0

Examples:

```
      100 100 DIV 10 0
10 0

      Y
14 16 28 30
18 20 32 35

      Y DIV 1 2 4 5
14  8  7  6
18 10  8  7
```

Program: *DIVRED* Group: *CALCULATE*

Syntax: *R*←*DIVRED X* Subroutine: *DIV*

Description: "Divide reduces" a two-row matrix of time series to a quotient time series. A zero denominator produces a zero quotient.

Input: *X* = Two-row matrix of time series

Output: *R* = Single time series quotient of *X*

Example:

```
      X
100 120 150 140 180
 25  20  30  28  30

      DIVRED X
4 6 5 5 6
```

Program: *GREATEST* Group: *CALCULATE*

Syntax: *R←GREATEST X* Subroutines: None

Description: Calculates the largest value in a time series, or the largest values in a matrix of time series.

Input: *X* = Vector or matrix

Output: *R* = Largest value in a vector or the row maxima of a matrix

Examples:

```
      GREATEST 23 16 37 19
37

      X
23 16 37 19
45 43 32 21
18 27 18 31

      GREATEST X
37 45 31
```

Program: *LEAST* Group: *CALCULATE*

Syntax: *R←LEAST X* Subroutines: None

Description: Calculates the smallest value in a time series, or the smallest values in a matrix of time series.

Input: *X* = Vector or matrix

Output: *R* = Smallest value in a vector or the row minima of a matrix

Examples:

```
      LEAST 23 16 37 19
16

      X
23 16 37 19
45 43 32 21
18 27 18 31

      LEAST X
16 21 18
```

Program: *MPROD* Group: *CALCULATE*

Syntax: *R←Y MPROD X* Subroutines: None

Description: Multiplication of a time series by several factors.

Input: *X* = Time series
Y = Vector of factors

Output: *R* = Matrix whose rows consist of the original time series multiplied by the respective factors.

Example:

```
      10 20 30 MPROD 1 2 3 4
10  20  30  40
20  40  60  80
30  60  90 120
```

Program: *TIMES* Group: *CALCULATE*

Syntax: *R←Y TIMES X* Subroutines: None

Description: Extends the multiplication primitive function to handle multiplication between vectors and matrices.

Input: *X* = Scalar, time series, or matrix
Y = Scalar, time series, or matrix

If neither *X* nor Y is a scalar, lengths must conform.

Output: *R* = Y×*X* (performed row by row when operating on a vector and matrix)

Example:

```
      Y
14 16 28 30
18 20 32 35

      Y TIMES 4 2 3 1
56 32 84 30
72 40 96 35
```

Program: *PCTOF* Group: *CALCULATE*

Syntax: *R*←Y *PCTOF X* Subroutine: *TIMES*

Description: Multiplies one array by an array of percents.

Input: *X* = Vector or matrix of data

Y = Array of percents: a scalar, a vector with as many elements as columns in *X*, or a matrix the same shape as *X*

Output: *R* = Data array resulting from multiplying *X* and *Y* and dividing by 100. A vector multiplied by a matrix is multiplied by each row of the matrix to produce a matrix result.

Examples:

```
      40 PCTOF 100 120 125
40 48 50

      X
100 120 150 140 180
150 170 110 140 180

      10 PCTOF X
10 12 15 14 18
15 17 11 14 18

      10 11 12 15 20 PCTOF X
10   13.2  18     21   36
15   18.7  13.2   21   36
```

Program: *PLUS* Group: *CALCULATE*

Syntax: *R*←Y *PLUS X* Subroutines: None

Description: Extends the addition primitive function to handle addition between vectors and matrices.

Input: *X* = Scalar, time series, or matrix
Y = Scalar, time series, or matrix

If neither *X* nor *Y* is a scalar, lengths must conform.

Output: *R* = *Y*+*X* (performed row by row when operating on a vector and matrix)

Example:

```
      Y
14 16 28 30
18 20 32 35

      Y PLUS 4 2 3 1
18 18 31 31
22 22 35 36
```

Program: *MINUS* Group: *CALCULATE*

Syntax: *R*←Y *MINUS X* Subroutines: None

Description: Extends the subtraction primitive function to handle subtraction between vectors and matrices.

Input: *X* = Scalar, time series, or matrix
Y = Scalar, time series, or matrix

If neither *X* nor *Y* is a scalar, lengths must conform.

Output: *R* = *Y*-*X* (performed row by row when operating on a vector and matrix)

Example:

```
      Y
14 16 28 30
18 20 32 35

      Y MINUS 4 2 3 1
10 14 25 29
14 18 29 34
```

Program: *TOTAL* Group: *CALCULATE*

Syntax: *R←TOTAL X* Subroutines: None

Description: Totals all the values in a time series or a matrix of time series.

Input: *X* = Time series or matrix whose rows are single time series.

Output: *R* = Single time series total if *X* is a single time series, or a vector of totals if *X* is a matrix of time series. *R* is equivalent to *+/X*.

Examples:

```
      X
0 20 30 40 50

      TOTAL X
150

      X
10 20 30 40 50
15 25 35 50 70
25 40 60 85 90

      TOTAL X
150 195 300
```

Program: *AGGREGATE* Group: *CALCULATE*

Syntax: *R←AGGREGATE X* Subroutines: None

Description: Produces a vector sum of a matrix of time series.

Input: *X* = Matrix of time series

Output: *R* = Sum of the columns of *X*; a single time series

Example:

```
      X
100 120 150 140 180
150 170 110 210 140
110 190 130 100 130

      AGGREGATE X
360 480 390 450 450
```

Program: *WTDSUM* Group: *CALCULATE*

Syntax: *R*←Y *WTDSUM X* Subroutines: None

Description: Calculates the weighted sum of a vector or matrix of data.

Input: *X* = Numeric vector or matrix

Y = Numeric vector of weights. If *X* is a vector, the lengths of *X* and *Y* must conform. If *X* is a matrix, then *Y* must have one weight for each row of *Y*.

Output: *R* = Weighted sum. If *X* is a vector, then *R* is

```
(Y[1]×X[1]) + ... + (Y[N]×X[N])
```

If *X* is a matrix, these linear combinations are performed row by row, as shown below.

```
(Y[1]×X[1;]) + ... + (Y[N]×X[N;])
```

Examples:

```
      2 ¯1 3 1 WTDSUM 10 15 12 20
61

      X
10 15 12 20
14 16 18 20
 5 10 15 20

      2 ¯1 3 WTDSUM X
21 44 51 80
```

Program: *YTD* Group: *CALCULATE*

Syntax: *R*←Y *YTD X* Subroutines: None

Description: Calculates year-to-date cumulative sums for each individual year in a time series.

Input: *X* = Vector time series or a matrix of time series with one time series per row

Y = Model periodicity
1 ←→ monthly
3 ←→ quarterly
6 ←→ half-yearly

Output: *R* = Cumulative sums for the data in each year, of the same shape as *X*

Examples:

```
      3 YTD 10 12 9 12 13 10 11 12
10 22 31 43 13 23 34 46

      X
1  2  3  4  5  6  7  8
9 10 11 12 13 14 15 16

      3 YTD X
1  3  6 10  5 11 18 26
9 19 30 42 13 27 42 58

      6 YTD X
1  3  3  7  5 11  7 15
9 19 11 23 13 27 15 31
```

Program: *BALFOR* Group: *CASH*

Syntax: *R←Y BALFOR X* Subroutines: None

Description: Calculates balance forward for balance sheet statistics.

Input: *X* = Changes to balance
Y = Opening balance

Output: *R* = Ending balances on hand

Example:
```
      20 BALFOR 10 ¯10 20 ¯20
30 20 40 20
```

Program: *SBALFOR* Group: *CASH*

Syntax: *R←Y SBALFOR X* Subroutines: None

Description: Calculates balance forward with initial balance preserved.

Input: *X* = Changes to balance
Y = Opening balance

Output: *R* = Closing balances on hand with *Y* as first amount

Example:
```
      20 SBALFOR 10 ¯10 20 ¯20
20 20 40 20
```


Program: *BREAKEVEN* Group: *CASH*

Syntax: *R←Y BREAKEVEN X* Subroutines: None

Description: Calculates the breakeven period given an opening balance and a cash flow.

Input: *X* = Cash flow time series or a matrix of cash flows, with one cash flow per row

Y = Opening balance(s) for the cash flow(s)

Y should be a scalar if *X* is a single cash flow, and a vector if *X* is a matrix of cash flows. If *Y* is not long enough, it is padded with zeros.

Output: *R* = Positive integers that designate the breakeven period for each cash flow. The breakeven period is the period when the opening balance plus the cumulative cash flow is first non-negative. If this value is always negative, one plus the length of the cash flow is returned.

Examples:

```
      F
¯20 ¯5 5 10 20 40 70

      10 BREAKEVEN F
4

      0 BREAKEVEN F
5

      ¯11 BREAKEVEN F
6

      CF
¯20 ¯5   5  10  20  40  70
¯25 10  10  10  10  10  10

      0 BREAKEVEN CF
5 4

      ¯12 ¯10 BREAKEVEN CF
6 5

      0 ¯50 BREAKEVEN CF
5 8
```

Program: *CURRENT* Group: *CASH*

Syntax: *R←Y CURRENT X* Subroutines: None

Description: Calculates the current and long term portion of debt for balance sheet statistics.

Input: *X* = Total debt principal payment stream
Y = Number of periods current (comprising 1 year)

Output: *R* = Current portion of debt
Z1 = Long term portion of debt (global variable)

Example:

```
      4 CURRENT 10 10 10 10 20 20 20 20
40 50 60 70 80 60 40 20
      Z1
80 60 40 20 0 0 0 0
```

Program: *PAYMT* Group: *CASH*

Syntax: *R←Y PAYMT X* Subroutine: *MQY*

Description: Calculates lumped payment amounts for cash flow detail.

Input: *X* = Payments incurred

Y = Periodicity and payment method
- ±1 ←→ monthly
- ±3 ←→ quarterly
- ±12 ←→ yearly
- \+ ←→ advance payment
- \- ←→ arrears payment

Output: *R* = Payments actually made

Example:

```
      ¯3 PAYMT 1 2 3 4 5 6
0 0 6 0 0 15
```

Program: *∆BAL* Group: *CASH*

Syntax: *R←Y ∆BAL X* Subroutines: None

Description: Calculates changes in balances for cash flow statement, and sources and uses of funds statement.

Input: *X* = Balances on hand for each period
Y = Opening balance

Output: *R* = Changes in balances

Example:
```
      20 ∆BAL 30 20 40 20
10 ¯10 20 ¯20
```

Program: *∆BALS* Group: *CASH*

Syntax: *R←Y ∆BALS X* Subroutines: None

Description: Calculates changes in balances (with specified first change) for cash flow, and sources and uses of funds statements.

Input: *X* = Balances on hand for each period
Y = Opening balance

Output: *R* = Changes in balances with *Y* as first term

Example:
```
      20 ∆BALS 30 20 40 20
20 ¯10 20 ¯20
```

Program: *MQY* Group: *CHANGETIME*

Syntax: *R*←Y *MQY X* Subroutines: None

Description: Converts one or more time series to a different periodicity.

Input: *X* = Original time series or matrix of time series

Y[1] = Original periodicity
- 1 ↔ monthly
- 3 ↔ quarterly
- 6 ↔ half-yearly
- 12 ↔ yearly

Y[2] = Desired periodicity

Y[3] = Optional -- Type of conversion (default = 0)
- 0 ↔ sum when compressing; divide into equal parts when expanding
- 1 ↔ average when compressing; replicate when expanding
- 2 ↔ take the first periodic value when compressing; interpolate linearly from the current value to the subsequent one when expanding
- 3 ↔ take the last periodic value when compressing; interpolate linearly from the previous value to the current one when expanding
- 4 ↔ take the first value when compressing; insert zeros after each number when expanding
- 5 ↔ take the last value when compressing; insert zeros before each number when expanding

Y[4-*N*] = Optional -- Closing values when Y[3]=2 and expanding; opening values when Y[3]=3 and expanding. One value is permitted for each time series being converted (default = 0).

Output: *R* = Converted time series or matrix of time series

Examples: In the following examples, *YTS* is a yearly time series, and *MQY* is used to convert *YTS* to a quarterly time series.

```
      YTS
100 200 400

      12 3 MQY YTS
25 25 25 25 50 50 50 50 100 100 100 100

      12 3 1 MQY YTS
100 100 100 100 200 200 200 200 400 400 400 400
```

In the following example, the closing value is 0.

```
      12 3 2 MQY YTS
100 125 150 175 200 250 300 350 400 300 200 100
```

In the following example, the closing value is 300.

```
      12 3 2 300 MQY YTS
100 125 150 175 200 250 300 350 400 375 350 325
```

In the following example, the opening value is 0.

```
      12 3 3 MQY YTS
25 50 75 100 125 150 175 200 250 300 350 400
```

In the following example, the opening value is 60.

```
      12 3 3 60 MQY YTS
70 80 90 100 125 150 175 200 250 300 350 400

      12 3 4 MQY YTS
100 0 0 0 200 0 0 0 400 0 0 0

      12 3 5 MQY YTS
0 0 0 100 0 0 0 200 0 0 0 400
```

In the following examples, *QTS* is a quarterly time series, and *MQY* is used to convert *QTS* to a yearly time series.

```
      QTS
10 20 30 40 50 60 70 80 90 100 110 120

      3 12 MQY QTS
100 260 420

      3 12 1 MQY QTS
25 65 105
```

The following example is equivalent to 3 12 4 *MQY QTS*.

```
      3 12 2 MQY QTS
10 50 90
```

The following example is equivalent to 3 12 5 *MQY QTS*.

```
      3 12 3 MQY QTS
40 80 120
```

Notes: For each choice of Y[3], the compression and expansion algorithms are inverses of each other. Conversion for a matrix of time series (one time series per row) works analogously: that is, the specified conversion method applies uniformly to all rows. The converted result is a matrix.

Program: *MQYFACTOR* Group: *CHANGETIME*

Syntax: *R←Y MQYFACTOR X* Subroutines: None

Description: Replicates data to reflect a change in model periodicity.

Input: *X* = Time series to be replicated

Y = Two-element vector of periodicities
 1 ↔ monthly
 3 ↔ quarterly
 12 ↔ yearly

Y[1] = Periodicity of input time series

Y[2] = Periodicity of output time series

Output: *R* = Adjusted time series where each term of *X* appears with frequency *Y*[1]÷*Y*[2] if *Y*[1]>*Y*[2]. If *Y*[1]<*Y*[2], the central term from each group of *Y*[2]÷*Y*[1] is selected.

Examples:

```
      3 1 MQYFACTOR 1 2 3 4 5 6 7 8
1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6 7 7 7 8 8 8

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

Program: *FIRST* Group: *CHANGETIME*

Syntax: *R←Y FIRST X* Subroutines: None

Description: Sets the starting position of data in a time span.

Input: *X* = Time series

Y[1] = Output time series length

Y[2] = One less than the starting position of input time series *X*

Output: *R* = Time series of length *Y*[1] containing *Y*[2] zeros and then remaining data from *X*

Example:
```
      6 3 FIRST 10 20 30 40 50
0 0 0 10 20 30
```

Program: *LAST* Group: *CHANGETIME*

Syntax: *R←Y LAST X* Subroutines: None

Description: Sets the ending position of data in a time span.

Input: *X* = Time series

Y[1] = Output time series length

Y[2] = One more than the ending position of input time series *X*

Output: *R* = Time series of length *Y*[1] containing *Y*[2]-1 terms of *X* with zeros for the remainder of the data.

Example:
```
      6 4 LAST 10 20 30 40 50
10 20 30 0 0 0
```

Program: *HALFYR* Group: *CHANGETIME*

Syntax: *R←Y HALFYR X* Subroutines: None

Description: Performs a half-year shift on a time series.

Input: *X* = Time series

Y = Periodicity
1 ←→ monthly
3 ←→ quarterly
12 ←→ yearly

Output: *R* = Shifted time series whose length has been extended so no data is lost

Examples:

```
      1 HALFYR 2 2 2 4 4 4 6 6 6 8 8 8
0 0 0 0 0 0 2 2 2 4 4 4 6 6 6 8 8 8

      3 HALFYR 2 2 2 4 4 4 6 6 6 8 8 8
0 0 2 2 2 4 4 4 6 6 6 8 8 8

      12 HALFYR 2 2 2 4 4 4 6 6 6 8 8 8
1 2 2 3 4 4 5 6 6 7 8 8 4
```

Program: *MTW* Group: *CHANGETIME*

Syntax: *R←Y MTW X* Subroutines: None

Description: Converts monthly data to weekly data.

Input: *X* = Monthly time series

Y = Vector of remaining workdays per month for the months corresponding to the monthly data

Output: *R* = Converted weekly data

Example:

```
      15 20 MTW 300 500
100 100 100 125 125 125 125
```

Program: *WTM* Group: *CHANGETIME*

Syntax: *R←Y WTM X* Subroutines: None

Description: Converts weekly data to monthly data.

Input: *X* = Weekly time series

Y = Vector of remaining workdays per month for the months corresponding to the weekly data

Output: *R* = Converted monthly data

Example:

```
      15 20 WTM 100 100 100 125 125 125 125
300 500
```

Program: *PRIOR* Group: *CHANGETIME*

Syntax: *R←PRIOR X* Subroutines: None

Description: Produces the values of a time series from the prior periods. When used in combination with *INIT*, generates an initial condition and subsequent terms.

Input: *X* = Time series

Output: *R* = Time series, where the last value of *X* is dropped, and a zero is added to the front

Examples:

```
      PRIOR 1 2 3 4 5
0 1 2 3 4

      25 INIT PRIOR 1 2 3 4 5
25 1 2 3 4
```

Program: *SHIFT* Group: *CHANGETIME*

Syntax: *R*←*Y* *SHIFT* *X* Subroutines: None

Description: Shifts data along a time axis.

Input: *X* = Time series
Y = Shift index

Output: *R* = Shifted time series. If *Y*>0, *Y* terms are dropped from the back of *X*, and *Y* zeros are added to the front. If *Y*<0, |*Y* terms are dropped from the front of *X*, and |*Y* zeros are added to the back.

Examples:

```
      2 SHIFT 1 2 3 4 5
0 0 1 2 3

      ¯2 SHIFT 1 2 3 4 5
3 4 5 0 0
```

Program: *CDCF* Group: *CONTRATES*

Syntax: *R*←Y *CDCF X* Subroutines: None

Description: Calculates continuous discount cash flow.

Input: *X* = Cash stream

Y[1] = Type of cash flow
- 1 ↔ in advance (payment at the beginning of each period)
- 0 ↔ continuous (payment spread evenly throughout each period)
- ¯1 ↔ in arrears (payment at the end of each period)

Y[2] = Periodicity
- 1 ↔ monthly
- 3 ↔ quarterly
- 6 ↔ half-yearly
- 12 ↔ yearly

Y[3 *TO N*] = Nominal annual discount rates, year by year. If enough rates are not provided, the last one is replicated as necessary.

Output: *R* = Discounted cash flow, where discount is compounded continuously

Z1 = 1 + the nominal periodic discount rates (global variable)

Examples:

```
      1 3 .08 .1 CDCF 100 100 100 100 100 100
100 98.02 96.079 94.176 92.312 90.032
      Z1
1.02 1.02 1.02 1.02 1.025 1.025

      0 3 .08 .1 CDCF 100 100 100 100 100 100
99.007 97.046 95.125 93.241 91.167 88.916
```

Z1 is the same as the value listed above.

```
      ¯1 3 .08 .1 CDCF 100 100 100 100 100 100
98.02 96.079 94.176 92.312 90.032 87.81
```

Z1 is the same as the value listed above.

Program: *CIROR* Group: *CONTRATES*

Syntax: *R←Y CIROR X* Subroutine: *CIDF*

Description: Calculates a continuous internal rate of return.

Input: *X* = Cash stream, including both revenues and expenses

Y[1] = Model periodicity
1 ←→ monthly
3 ←→ quarterly
6 ←→ half-yearly
12 ←→ yearly

Y[2] = Optional -- Type of rate when periodicity is not yearly
0 ←→ produce the effective annual rate (default)
1 ←→ produce the nominal annual rate

Output: *R* = The annual continuous internal rate of return of the cash stream. No reinvestment of positive stream benefits is assumed; *R* equals the rate needed to yield a net present value (with continuous compounding) of zero. With multiple rates of return, the rate closest to 10% is calculated.

As long as compounding is continuous, the rates of return are the same even if the flows are continuous within the periods or occur at the ends of the periods.

Example:

```
      12 CIROR ¯300 150 100 200 50 ¯100 50 100
.2350974436
```

Program: *CIDF* Group: *CONTRATES*

Syntax: *R*←Y *CIDF X* Subroutines: None

Description: Calculates a continuous internal discount factor.

Input: *X* = Cash stream, including both revenues and expenses

Y = Initial approximation of the internal discount factor

Output: *R* = The internal discount factor of the cash stream. No reinvestment of positive stream benefits is assumed; the result is the discount factor needed to yield a net present value (with continuous compounding) of zero. With multiple factors, the factor closest to .9 is calculated.

Discount factor = 1÷(1+rate of return).

Example:
```
      .9 CIDF ¯300 150 100 200 50 ¯100 50 100
.8096527162
```

Program: *CFV* Group: *CTIMEVALUE*

Syntax: *R←Y CFV X* Subroutine: *CDCF*

Description: Calculates continuous future value of a cash stream.

Input: *X* = Cash stream

Y[1] = Type of cash flow
- 1 ←→ in advance (payment at the beginning of each period)
- 0 ←→ continuous (payment spread evenly throughout each period)
- ¯1 ←→ in arrears (payment at the end of each period)

Y[2] = Periodicity
- 1 ←→ monthly
- 3 ←→ quarterly
- 6 ←→ half-yearly
- 12 ←→ yearly)

Y[3 *TO N*] = Nominal annual discount rates, year by year. If enough rates are not provided, the last one is replicated as necessary.

Output: *R* = Total future value, with discount rates compounded continuously.

Examples:

```
      1 3 .08 .1 CFV 100 100 100 100 100 100
648.9256095

      0 3 .08 .1 CFV 100 100 100 100 100 100
641.9686924

      ¯1 3 .08 .1 CFV 100 100 100 100 100 100
635.0622527
```

Program: *CPV* Group: *CTIMEVALUE*

Syntax: *R*←Y *CPV X* Subroutine: *CDCF*

Description: Calculates continuous present value of a cash stream.

Input: *X* = Cash stream

```
Y[1] = Type of cash flow
         1 ←→ in advance (payment at the beginning of
               each period)
         0 ←→ continuous (payment spread evenly
               throughout each period)
        ¯1 ←→ in arrears (payment at the end of each
               period)

Y[2] = Periodicity
         1 ←→ monthly
         3 ←→ quarterly
         6 ←→ half-yearly
        12 ←→ yearly

Y[3 TO N] = Nominal annual discount rates, year by
            year.  If enough rates are not provided,
            the last one is replicated as necessary.
```

Output: *R* = Total present value, with discount rates compounded continuously.

Examples:

```
      1 3 .08 .1 CPV 100 100 100 100 100 100
570.6193515

      0 3 .08 .1 CPV 100 100 100 100 100 100
564.5019299

      ¯1 3 .08 .1 CPV 100 100 100 100 100 100
558.4288946
```

Program: *EPRIN* Group: *DEBT*

Syntax: *R←EPRIN X* Subroutines: None

Description: Calculates equal principal payment loan data.

Input: *X*[1] = Periodicity
1 ←→ monthly
3 ←→ quarterly
12 ←→ yearly

X[2] = Principal amount

X[3] = Number of periods

X[4] = Annual interest rate

X[5] = Optional -- Payment method
0 ←→ arrears (default)
1 ←→ advance

Output: *R* = Periodic interest payments
Z1 = Periodic principal payments (global variable)

Examples:

```
      EPRIN 12 1000 4 .05
50 37.5 25 12.5
      Z1
250 250 250 250

      EPRIN 12 1000 4 .05 1
37.5 25 12.5 0
      Z1
250 250 250 250
```

Program: *GLOAN* Group: *DEBT*

Syntax: *R←Y GLOAN X* Subroutines: None

Description: Calculates loan data given payment schedule.

Input: *X*[1] = Periodicity
1 ←→ monthly
3 ←→ quarterly
12 ←→ yearly

X[2] = Principal amount

X[3] = Number of periods

X[4] = Annual interest rate

X[5] = Optional -- Payment method
0 ←→ arrears (default)
1 ←→ advance

Y = Annuity (principal plus interest) payments

Output: *R* = Periodic interest payments
Z1 = Periodic principal payments (global variable)

If an outstanding balance remains at the end of the specified periods, a final balloon payment is added.

Examples:

```
      100 500 200 GLOAN 12 1000 3 .05
50 47.5 24.875 16.11875
      Z1
50 452.5 175.125 322.375

      100 500 200 GLOAN 12 1000 3 .05 1
47.3684 23.5457 14.2586 0
      Z1
52.6316 476.4543 185.7414 285.1728
```

Program: *LEVPAY* Group: *DEBT*

Syntax: *R←LEVPAY X* Subroutines: None

Description: Calculates level payment loan data.

Input: *X*[1] = Periodicity
 1 ←→ monthly
 3 ←→ quarterly
 6 ←→ half-yearly
 12 ←→ yearly

X[2] = Principal amount

X[3] = Number of periods

X[4] = Annual interest rate

X[5] = Optional -- Payment method
 0 ←→ arrears (default)
 1 ←→ advance

Output: *R* = Periodic interest payments
Z1 = Periodic principal payments (global variable)

Examples:

```
      LEVPAY 12 1000 4 .05
50 38.3994 26.2188 13.4291
      Z1
232.0118 243.6124 255.7930 268.5827

      LEVPAY 12 1000 4 .05 1
38.4447 26.2812 13.4775 0
      Z1
231.1062 243.2697 256.0733 269.5509
```

Program: *DLEVPAY* Group: *DEBT*

Syntax: *R←DLEVPAY X* Subroutine: *IDF* (if *X*[4]=0)

Description: Derives level payment loan (mortgage) parameters; the input to the *LEVPAY* program.

Input: *X* is a five-element numeric vector where exactly one of *X*[2] through *X*[5] is zero, and all the other elements are positive. *DLEVPAY* solves for the item represented by the zero using the other elements of *X*.

X[1] = Periodicity
- 1 ↔ monthly
- 3 ↔ quarterly
- 6 ↔ half-yearly
- 12 ↔ yearly

X[2] = Principal amount of the loan

X[3] = Number of periods (rounded up to an integer)

X[4] = Nominal annual interest rate (as a decimal)

X[5] = Periodic annuity (principal+interest) payment

Output: *R* is the same as *X*, except that the zero becomes the desired amount. To obtain a period-by-period breakdown of the principal and interest, *R*[1 2 3 4] can be used as the argument of *LEVPAY*. *DLEVPAY* assumes that the payment method is "in arrears".

Examples: In the following example, *DLEVPAY* finds the amount that can be borrowed at 12% interest with a $177 payment over 10 years.

```
      DLEVPAY 12 0 10 .12 177
12 1000.089476 10 0.12 177
```

In this example, *DLEVPAY* finds the number of years it will take to repay $1000 at 12% interest, with a yearly payment of $177.

```
      DLEVPAY 12 1000 0 .12 177
12 1000 10 0.12 177
```

In this example, *DLEVPAY* finds the interest rate being charged on a $1000 loan, with a ten-year payment schedule of $177 per year.

```
      DLEVPAY 12 1000 10 0 177
12 1000 10 0.120021858 177
```

In this example, *DLEVPAY* finds the yearly payment on a $1000 loan borrowed at 12% interest.

```
      DLEVPAY 12 1000 10 .12 0
12 1000 10 0.12 176.9841642
```

See the documentation for *LEVPAY* for more information on the following example.

```
      INT←LEVPAY 12 1000 10 .12        First year
                                       breakout

      INT[1]                           Interest
120

      Z1[1]                            Principal
56.9841642

      INT[1]+Z1[1]                     Total payment
176.9841642
```

Program: *LEVPAY1* Group: *DEBT*

Syntax: *R*←Y *LEVPAY1 X* Subroutines: *LEVPAY*, *ROUND*

Description: Calculates approximately level-payment loan data when the interest rate varies from year to year, or when the principal portion of the payment must be a multiple of a given unit.

Input: *X* = Vector of interest rates, one per year

Y[1] = Principal loan amount

Y[2] = Optional -- Periodicity of interest and payment schedule (default = 1)
- 1 ↔ monthly
- 3 ↔ quarterly
- 6 ↔ half-yearly
- 12 ↔ yearly

Y[3] = Optional -- Unit amount of which each principal payment is a multiple (default = 1)

Y[4] = Optional -- Iteration tolerance for the payment average (default = .1×Y[3])

Y[5] = Optional -- Limit on the number of iterations (default = 10)

Output: *R* = Periodic stream of interest payments

Z1 = Periodic stream of principal payments (global variable)

Z2 = Periodic stream of total payments (*R*+Z1) (global variable)

Examples: In the following examples, *X* is the annual interest rates for four years.

```
      X
.05 .1 .08 .05

      10000 12 100 LEVPAY1 X     Multiples of $100
500 750 424 140
      Z1
2500 2200 2500 2800
      Z2
3000 2950 2924 2940

      10000 12 50 LEVPAY1 X      Multiples of $50
500 755 428 140
      Z1
2450 2200 2550 2800
      Z2
2950 2955 2978 2940
```

```
      10000 6 50 LEVPAY1 X   Half-yearly payments
250 220 377.5 325 214 164 70 35
      Z1
1200 1250 1050 1150 1250 1300 1400 1400
      Z2
1450 1470 1427.5 1475 1464 1464 1470 1435
```

Notes: *LEVPAY1* uses an iterative technique to find a sequence of principal payments where the total payment is approximately constant. Iteration halts when either the difference between successive total payment averages is less than the specified iteration tolerance, or until the specified number of iterations are run. The length of the result (and of *Z1* and *Z2*) depends on the number of years (the length of *X*) and the periodicity. Any adjustment to the principal stream necessitated by rounding is made in the final period. The payment type is assumed to be in arrears; that is, interest and principal are paid at the end of each period.

Program: *LOAN* Group: *DEBT*

Syntax: *R←LOAN X* Subroutines: *LEVPAY*, *EPRIN*, *TLOAN*

Description: Calculates selectable loan data.

Input: *X*[1] = Loan type
- 1 ↔ level payment
- 2 ↔ equal principal
- 3 ↔ term loan

X[2] = Periodicity
- 1 ↔ monthly
- 3 ↔ quarterly
- 12 ↔ yearly

X[3] = Principal amount

X[4] = Number of periods

X[5] = Annual interest rate

X[6] = Optional -- Payment method
- 0 ↔ arrears (default)
- 1 ↔ advance

Output: *R* = Periodic interest payments
Z1 = Periodic principal payments (global variable)

Examples: See the examples for programs *LEVPAY*, *EPRIN*, and *TLOAN*.

Program: *LOFC* Group: *DEBT*

Syntax: *R←Y LOFC X* Subroutines: None

Description: Performs line-of-credit calculations.

Input: *X* = Cash stream (negative numbers are loan takedowns).

Y[1] = Periodicity and payment type
- ±1 ↔ monthly
- ±3 ↔ quarterly
- ±12 ↔ yearly
- \+ ↔ payments in advance
- \- ↔ payments in arrears

Y[2] = Credit limit

Y[3] = Annual interest rate

Y[4] = Placement fee percentage (as a decimal)

Y[5] = Optional -- Contingency fee percentage (as a decimal; default = 0)

Output: *R* = Interest costs
Z1 = Principal payments (global variable)
Z2 = Fees and acquisition costs (global variable)
Z3 = Loan balances (global variable)

Notes: For every negative flow in the cash stream, money is borrowed up to the specified credit limit. Interest is paid on the unpaid balance, and a placement fee at the specified percentage rate is collected on each loan. The contingency fee percentage value is the rate paid on money committed but not used; that is, credit limit minus current loan balance.

Examples:

```
      12 10000 .08 .01 .005 LOFC ¯2000 ¯1000 0
   1500 3000
160 240 240 120 0
      Z1
0 0 0 1500 1500
      Z2
60 45 35 42.5 50
      Z3
2000 3000 3000 1500 0

      ¯12 10000 .08 .01 .005 LOFC ¯2000 ¯1000 0
   1500 3000
0 160 240 240 120
      Z1
0 0 0 0 1500
      Z2
50 60 45 35 42.5
      Z3
0 2000 3000 3000 1500
```

Program: *TLOAN* Group: *DEBT*

Syntax: *R←TLOAN X* Subroutines: None

Description: Calculates term loan data.

Input: *X*[1] = Periodicity
 1 ↔ monthly
 3 ↔ quarterly
 12 ↔ yearly

X[2] = Principal amount

X[3] = Number of periods

X[4] = Annual interest rate

X[5] = Optional -- Payment method
 0 ↔ arrears (default)
 1 ↔ advance

Output: *R* = Periodic interest payments
Z1 = Periodic principal payments (global variable)

Examples:

```
      TLOAN 12 1000 4 .05
50 50 50 50
      Z1
0 0 0 1000

      TLOAN 12 1000 4 .05 1
50 50 50 0
      Z1
0 0 0 1000
```

Program: *VLOAN* Group: *DEBT*

Syntax: R←Y *VLOAN* X Subroutines: None

Description: Calculates loan data for a variable payment schedule.

Input: X[1] = Periodicity
 1 ↔ monthly
 3 ↔ quarterly
 12 ↔ yearly

X[2] = Principal amount

X[3] = Number of periods

X[4] = Annual interest rate

X[5] = Optional -- Payment method
 0 ↔ arrears (default)
 1 ↔ advance

Y= Principal payments

Output: R = Periodic interest payments
Z1 = Periodic principal payments (global variable)

If there is an outstanding balance at the end of the specified number of periods, a final balloon payment is added.

Examples:

```
      100 500 200 VLOAN 12 1000 3 .05
50 45 20 10
      Z1
100 500 200 200

      100 500 200 VLOAN 12 1000 3 .05 1
45 20 10 0
      Z1
100 500 200 200
```

Program: *ACRS* Group: *DEPRECIATION*

Syntax: *R←Y ACRS X* Subroutines: None

Description: Calculates the total depreciation stream produced by a stream of capital investments, using the Accelerated Cost Recovery System (ACRS) method.

Input: *X* = Time series of capital investments

Y[1] = ACRS class type (3, 5, 10, or 15)

Y[2] = Optional -- Starting year for the time series (if 0 or omitted, the current year (⎕*TS*[1]) is used)

Y[3] = Optional -- Periodicity of the time series (if 0 or omitted, yearly is assumed)
 1 ←→ monthly
 3 ←→ quarterly
 6 ←→ half-yearly
12 ←→ yearly

Output: *R* = Time series of all the depreciation produced by *X*. The depreciation produced by each element of *X* is accumulated into the proper elements of *R*. The periodicity of *R* is the same as that of *X*. (Use the program *MQY* to change the periodicity of the result.)

Examples: In the following examples, *X* represents five expenditures.

```
      X
2000 4000 4000 2000 4000
```

In the following examples, the percents 25 38 37 are used for years 1983 and 1984, the percents 29 47 24 are used for 1985, and percents 33 45 22 are used for 1986 and 1987.

```
      3 ACRS X     Starting year = 1983 ←→ ⎕TS[1]
500 1760 3420 4020 3180 2240 880

      3 1984 ACRS X     Starting year = 1984
500 1920 3940 3420 3100 2240 880

      3 1985 ACRS X     Starting year = 1985
580 2260 3600 3340 3100 2240 880

      3 1986 ACRS X     Starting year = 1986
660 2220 3560 3340 3100 2240 880
```

The following example calculates quarterly expenditure starting in 1984.

```
      3 1984 3 ACRS X
125 375 625 750 1105 1235 1365 1430 1605 1595
      1585 1580 1165 795 425 240
```

The following examples use a 5-year recovery period, a 10-year recovery period, and a 15-year recovery period, respectively.

```
      5 1983 ACRS X
300 1040 2020 2980 3700 3240 1600 800 320

      10 1983 ACRS X
160 600 1160 1640 2000 2200 1980 1740 1540 1340
      960 400 200 80

      15 1983 ACRS X
100 400 820 1140 1460 1580 1500 1360 1260 1160
      1060 960 860 800 700 500 200 100 40
```

Notes: Under the ACRS method, one set of yearly percents is used for the years 1981 to 1984, another set for 1985, and a third set for years beginning 1986. (The ACRS method is not applicable for years prior to 1981.) The program determines the appropriate set of percents to use for each element of *X* from the starting year and the length and periodicity of *X*.

Program: *DEPRE* Group: *DEPRECIATION*

Syntax: *R←DEPRE X* Subroutines: *ACCSF*, *DBAL*, *SFUND* (calling *LEVPAY*) *STL*, *SYD*

Description: Calculates selectable depreciation methods.

Input: *X*[1] = Depreciation type
- 1 ←→ straight line
- 2 ←→ sum of years digits
- 3 ←→ declining balance
- 4 ←→ sinking fund
- 5 ←→ accelerated sinking fund

X[2 *TO N*] = Input parameters for each depreciation method

Output: *R* = Monthly depreciation stream

Examples: For descriptions of the input parameters and examples of the computations, see the documentation for the programs *ACCSF*, *DBAL*, *SFUND*, *STL*, and *SYD*.

Program: *STL* Group: *DEPRECIATION*

Syntax: *R←STL X* Subroutines: None

Description: Calculates straight line depreciation.

Input: *X*[1] = Original basis

X[2] = Number of years of useful life (if fractional, the value is reduced internally to whole months)

X[3] = Salvage value (either a fraction of the original basis or an actual dollar amount)

Output: *R* = Monthly depreciation stream; one element for each month in the useful life

Examples: *MQY* is used here to convert monthly data to yearly data.

```
      D←STL 20000 5 2000
      ρD
60
      1 12 MQY D
3600 3600 3600 3600 3600

      1 12 MQY STL 20000 5 .1
3600 3600 3600 3600 3600

      D←STL 20000 4.5 2000
      ρD
54
      1 12 MQY D
4000 4000 4000 4000 2000
```

Program: *SYD* Group: *DEPRECIATION*

Syntax: *R←SYD X* Subroutines: None

Description: Calculates sum-of-years digits depreciation.

Input: *X*[1] = Original basis

X[2] = Number of years of useful life (if fractional, the value is reduced internally to whole months)

X[3] = Salvage value (either a fraction of the original basis or an actual dollar amount)

X[4] = Optional -- Year to force a switch to straight line depreciation

Output: *R* = Monthly depreciation stream, one element for each month in the useful life

Examples: *MQY* is used here to convert monthly data to yearly data.

```
      D←SYD 20000 5 2000
      ⍴D
60
      1 12 MQY D
6000 4800 3600 2400 1200

      D←SYD 20000 5 .1 3
      ⍴D
60
      1 12 MQY D
6000 4800 2400 2400 2400

      D←SYD 20000 4.5 2000
      ⍴D
54
      1 12 MQY D
6480 5040 3600 2160 720

      D←SYD 20000 4.5 2000 3
      ⍴D
54
      1 12 MQY D
6480 5040 2592 2592 1296
```

Program: *DBAL* Group: *DEPRECIATION*

Syntax: *R←DBAL X* Subroutines: None

Description: Calculates declining balance depreciation with or without an optimal switch.

Input: *X*[1] = Original basis

X[2] = Number of years of useful life (if fractional, the value is reduced internally to whole months)

X[3] = Salvage value (either a fraction of the original basis or an actual dollar amount)

X[4] = Percent declining, usually 200, 150, or 125

X[5] = Optional -- If equal to 1, indicates an optimal switch; if greater than 1, indicates a forced switch in the specified year

X[6] = Optional -- If equal to 1, requests a switch to straight line; if omitted or not equal to 1, requests a switch to sum of years digits

Output: *R* = Monthly depreciation stream

Examples: *MQY* is used here to convert monthly data to yearly data.

```
      1 12 MQY DBAL 10000 10 1000 200
2000 1600 1280 1024 819.2 655.4 524.3 419.4 335.5
      342.2

      1 12 MQY DBAL 10000 10 1000 200 1
2000 1600 1280 1030 882.9 735.7 568.6 441.4 294.3
      147.1

      1 12 MQY DBAL 10000 10 1000 200 5
2000 1600 1280 1024 884.6 737.1 589.7 442.3 294.9
      147.4

      1 12 MQY DBAL 10000 10 1000 200 1 1
2000 1600 1280 1024 819.2 655.4 524.3 419.4
      338.9 338.9

      1 12 MQY DBAL 10000 10 1000 200 5 1
2000 1600 1280 1024 516 516 516 516 516 516
```

DBAL returned 120 elements in the cases above, and 114 elements in the cases below.

```
      1 12 MQY DBAL 10000 9.5 1000 200
2105.3 1662.1 1312.1 1035.9 817.8 645.7 509.7
      402.4 317.7 191.3
```

```
1 12 MQY DBAL 10000 9.5 1000 200 1
2105.3 1662.1 1312.1 1040.1 880.1 720.1 560.1
400.1 240.0 80.0

1 12 MQY DBAL 10000 9.5 1000 200 1 1
2105.3 1662.1 1312.1 1035.9 817.8 645.7 509.7
402.4 339.4 169.7
```

Program: *SFUND* Group: *DEPRECIATION*

Syntax: *R←SFUND X* Subroutine: *LEVPAY*

Description: Calculates the sinking fund method of depreciation. Given an implied interest rate, the depreciation for each period is the same as the principal payment made on a level payment loan ("backend loaded") that would amortize the total depreciation.

Input: *X*[1] = Original basis

X[2] = Number of years of useful life

X[3] = Salvage value (either a fraction of the original basis or an actual dollar amount)

X[4] = Implied interest rate

Output: *R* = Monthly depreciation stream

Example: *MQY* is used here to convert monthly data to yearly data.

```
1 12 MQY SFUND 1000 5 100 .06
159.0217 168.8805 179.3504 190.4695 202.2779
```

Program: *ACCSF* Group: *DEPRECIATION*

Syntax: *R←ACCSF X* Subroutines: None

Description: Calculates the accelerated sinking fund method of depreciation where the depreciation balance is reduced at the compound rate necessary to amortize the total depreciation.

Input: *X*[1] = Original basis

X[2] = Number of years of useful life

X[3] = Salvage value (either a fraction of the original basis or an actual dollar amount)

Output: *R* = Monthly depreciation stream

Example: *MQY* is used here to convert monthly data to yearly data.

```
      1 12 MQY ACCSF 1000 5 100
369.0427 232.8502 146.9185 92.6993 58.4893
```

Program: *LEADLAG* Group: *DISTRIBUTE*

Syntax: *R*←Y *LEADLAG X* Subroutines: None

Description: Lead or lag a time series by a multiplier.

Input: *X* = Time series

Y[1] = Time index
Y[2 *TO N*] = Multiplicative factors

Output: *R* = Adjusted time series scaled by the factors and shifted by the time index. Generally, an input *X*[*I*] in a particular time period starts producing output Y[1] periods later, if Y[1]≥0; or |Y[1] periods earlier, if Y[1]<0. The successive outputs produced by *X*[*I*] are *X*[*I*]×Y[2], *X*[*I*]×Y[3], ..., *X*[*I*]×Y[*N*]. The length of *R* equals the length of *X*.

Examples:

```
      0 .6 .3 .1 LEADLAG 10 20 30 40 50
6 15 25 35 45

      1 .6 .3 .1 LEADLAG 10 20 30 40 50
0 6 15 25 35
```

Program: *DAYLAG* Group: *DISTRIBUTE*

Syntax: *R←DAYLAG X* Subroutines: None

Description: Calculates a lead lag operator from the number of days lead or lag.

Input: *X*[1] = Model periodicity
1 ↔ monthly
3 ↔ quarterly
12 ↔ yearly

X[2] = Number of days lead (-) or lag (+)

If *X* contains a single number, it is assumed to be *X*[2] and *X*[1] is assumed to be 1.

Output: *R* = Lead lag operator -- can be used as the left argument to the *LEADLAG* program.

Examples:

```
      DAYLAG 12 45
0 .875 .125

      DAYLAG 12 ¯45
¯1 .125 .875

      DAYLAG 45
1 .5 .5

      DAYLAG ¯45
¯2 .5 .5
```

Program: *QFACTOR* Group: *DISTRIBUTE*

Syntax: *R←Y QFACTOR X* Subroutines: None

Description: Multiplies quarterly factors across a monthly time series.

Input: *X* = Monthly time series

Y = Vector of quarterly multiplicative factors to be distributed across *X*

Output: *R* = Adjusted monthly time series where the first three terms of *X* are multiplied by Y[1], the next three terms of *X* are multiplied by Y[2], and so on.

Example:

```
      2 3 4 QFACTOR 1 2 3 1 2 3 1 2
2 4 6 3 6 9 4 8
```

Program: *YFACTOR* Group: *DISTRIBUTE*

Syntax: *R←Y YFACTOR X* Subroutines: None

Description: Multiplies yearly factors across a monthly time series.

Input: *X* = Monthly time series

Y = Vector of yearly multiplicative factors to be distributed across *X*

Output: *R* = Adjusted monthly time series where the first 12 terms of *X* are multiplied by Y[1], the next 12 terms of *X* are multiplied by Y[2], and so on.

Example:

```
      2 3 4 YFACTOR 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
   1 2 3 4 5 1 2 3 4 5
2 4 6 8 10 2 4 6 8 10 2 4 9 12 15 3 6 9 12 15 3 6
      9 12 20
```

Program: *SPREAD* Group: *DISTRIBUTE*

Syntax: *R←Y SPREAD X* Subroutines: None

Description: Spreads one time series across another.

Input: *X* = Time series
Y = Time series

Output: *R* = Time series which is the sum of the following vectors:

```
Y[1] × X,0 0 0 ... 0
Y[2] × 0,X,0 0 ... 0
Y[3] × 0 0,X,0 ... 0
.
.
.
Y[M] × 0 0 0 ... 0,X
```

Example:

```
      10 20 30 SPREAD .3 .4 .3 .2
3 10 20 20 13 6
```

Notes: The length of *R* is one less than the sum of the lengths of *X* and *Y*.

```
(Y SPREAD X) = X SPREAD Y
```

The relationship between *SPREAD* and *LEADLAG* is that the first (⍴*X*) terms of *Y SPREAD X* are the same as

```
(0,Y) LEADLAG X
```

The *K*th term of *Y SPREAD X* can be viewed as the sum of all products *X*[*I*]×*Y*[*J*] where (*I*+*J*)=*K*+1. For example,

```
R[3] = (X[1]×Y[3])+(X[2]×Y[2])+(X[3]×Y[1])
```

SPREAD can be used to calculate a depreciation time series; the input time series are the capital investment stream and the depreciation schedule for $1.

Program: *EYIR* Group: *INTEREST*

Syntax: *R←Y EYIR X* Subroutines: None

Description: Calculates effective yearly interest rates, given nominal rates.

Input: *X* = Nominal yearly interest rates

Y = Number of compoundings per year -- can be a vector matching the length of *X*; 0 indicates continuous compounding

Output: *R* = Effective yearly interest rates

Example:

```
      1 12 12 365 0 EYIR .06 .06 .07 .07 .07
.06 .06167781186 .07229008086 .07250098317
      .07250818125
```

Program: *NYIR* Group: *INTEREST*

Syntax: *R←Y NYIR X* Subroutines: None

Description: Calculates nominal yearly interest rates, given effective rates.

Input: *X* = Effective yearly interest rates

Y = Number of compoundings per year -- can be a vector matching the length of *X*; 0 indicates continuous compounding

Output: *R* = Nominal yearly interest rates

Example:

```
      1 12 12 365 0 NYIR .06 .06 .07 .07 .07
.06 .05841060678 .06784974465 .06766491967
      .06765864847
```

Program: *QTR2NDINTEREST* Group: *INTEREST*

Syntax: *R←Y QTR2NDINTEREST X* Subroutines: None

Description: Calculates secondary interest effects from a cash flow.

Input: *X* = Quarterly cash flows, unadjusted for the effects of marginal investing or borrowing

Y[1] = Effective tax rate
Y[2] = Annual interest rate on investments
Y[3] = Annual interest rate on loans
Y[4] = Opening cash balance (positive for investments, negative for loans)

Output: *R* = Net to cash after taxes

Z1 = Taxes paid (+) or tax reduction (-) as a result of investing or borrowing (global variable)

Z2 = Investments (+) or loans (-) after adjustments (global variable)

Z3 = Before tax revenue (+) or expense (-) (global variable)

When cash on hand is positive, marginal cash flow *R* is from investments; when cash on hand is negative, marginal cash flow *R* is a result of loans.

Taxes (Z1) are computed and "paid" quarterly; interest is computed monthly.

Example:

```
      .48 .08 .12 ¯1200 QTR2NDINTEREST 500 500 500
   500 500
¯15.5 ¯8.2 ¯1 4.5 9.6
      Z1
¯14.3 ¯7.6 ¯1 4.1 8.8
      Z2
¯715.5 ¯223.7 275.3 779.7 1289.3
      Z3
¯29.7 ¯15.8 ¯2 8.6 18.4
```


Program: *YR2NDINTEREST* Group: *INTEREST*

Syntax: *R←Y YR2NDINTEREST X* Subroutines: None

Description: Calculates secondary interest effects from a cash flow.

Input: *X* = Yearly cash flows, unadjusted for the effects of marginal investing or borrowing

Y[1] = Effective tax rate
Y[2] = Annual interest rate on investments
Y[3] = Annual interest rate on loans
Y[4] = Opening cash balance (positive for investments, negative for loans)

Output: *R* = Net to cash after taxes

Z1 = Taxes paid (+) or tax reduction (-) as a result of investing or borrowing (global variable)

Z2 = Investments (+) or loans (-) after adjustments (global variable)

Z3 = Before tax revenue (+) or expense (-) (global variable)

When cash on hand is positive, marginal cash flow *R* is from investments; when cash on hand is negative, marginal cash flow *R* is a result of loans.

Taxes (Z1) are computed and "paid" quarterly: interest is computed monthly.

Example:

```
      .48 .08 .12 ¯1200 YR2NDINTEREST 500 500 500
   500 500
¯59.5 ¯32.6 ¯4.6 17.7 38.9
      Z1
¯54.9 ¯30.1 ¯4.3 16.3 35.9
      Z2
¯759.5 ¯292.1 203.3 721 1259.9
      Z3
¯114.4 ¯62.6 ¯8.9 34 74.8
```

Program: *AVGCOST* Group: *INVENTORY*

Syntax: *R←Y AVGCOST X* Subroutines: None

Description: Calculates cost of goods sold on an average cost basis.

Input: *X*[1 *TO N*] = Total inventoried cost for *N* periods

X[*N*+1 *TO* 2*N*] = Total number of units produced for sale in the *N* periods

X[2*N*+1 *TO* 3*N*] = Total unit sales in each of the *N* periods

Y[1] = Opening unit stock

Y[2] = Total cost of the opening stock

Output: *R* = Cost of goods sold in each of the *N* periods

Example: In this example, *N*=2.

```
      100 250 AVGCOST 125 175 75 100 100 150
214.286 287.755
```

Program: *FIFO* Group: *INVENTORY*

Syntax: *R*←Y *FIFO X* Subroutines: None

Description: Calculates cost of goods sold on an first-in, first-out basis.

Input: *X*[1 *TO N*] = Total inventoried cost for *N* periods

X[*N*+1 *TO* 2*N*] = Total number of units produced for sale in the *N* periods

X[2*N*+1 *TO* 3*N*] = Total unit sales in each of the *N* periods

Y[1] = Opening unit stock

Y[2] = Total cost of the opening stock

Output: *R* = Cost of goods sold in each of the *N* periods

Example: In this example, *N*=2.

```
      100 250 FIFO 125 175 75 100 100 150
250 256.25
```

Program: *LIFO* Group: *INVENTORY*

Syntax: *R*←Y *LIFO X* Subroutines: None

Description: Calculates cost of goods sold on an last-in, first-out basis.

Input: *X*[1 *TO N*] = Total inventoried cost for *N* periods

X[*N*+1 *TO* 2*N*] = Total number of units produced for sale in the *N* periods

X[2*N*+1 *TO* 3*N*] = Total unit sales in each of the *N* periods

Y[1] = Opening unit stock

Y[2] = Total cost of the opening stock

Output: *R* = Cost of goods sold in each of the *N* periods

Example: In this example, *N*=2.

```
      100 250 LIFO 125 175 75 100 100 150
187.5 300
```

Program: *DISCOUNT*　　　　Group: *INVENTORY*

Syntax: *R←Y DISCOUNT X*　　　　Subroutines: None

Description: Calculates volume discount.

Input: *X* = Time series of dollar purchase amounts

Y = Discount schedule; that is, *Y*[odd] is the minimum amount of purchase necessary to earn a discount at the rate *Y*[odd+1]. Discount rates (the even elements of *Y*) are expressed as decimals.

Output: *R* = Time series of discounted amounts
Z1 = Actual discounts for each purchase (global variable)

Example:

```
      1000 .02 5000 .05 10000 .1 30000 .2 DISCOUNT
   500 1500 6000 20000
500 1470 5700 18000
      Z1
0 30 300 2000
```

Program: *VOLCOST*　　　　Group: *INVENTORY*

Syntax: *R←Y VOLCOST X*　　　　Subroutines: None

Description: Calculates total cost as a function of volume.

Input: *X* = Time series of unit demands

Y = Schedule of unit costs dependent on volume. *Y*[odd] is the minimum volume that can be bought at the price *Y*[odd+1].

Output: *R* = Total cost for each volume

Example:

```
      0 6.5 1000 6 5000 5.5 10000 5 VOLCOST 500
   1500 6000 20000
3250 9000 33000 100000
```

Program: *CORI* Group: *MANIPULATE*

Syntax: *R*←Y *CORI X* Subroutines: None

Description: Selects time series of calculated or input values.

Input: *X* = Time series of calculated values

Y = Time series of the same length as *X*, representing input values

Output: *R* = Time series defined as follows:
when Y≠0, *R*=Y
when Y=0, *R*=corresponding calculated value *X*

Example:
```
      10 0 30 0 50 CORI 12 14 16 18 20
10 14 30 18 50
```

Program: *ELTOF* Group: *MANIPULATE*

Syntax: *R*←Y *ELTOF X* Subroutines: None

Description: Selects elements of a time series.

Input: *X* = Time series
Y = Element numbers to be selected

Output: *R* = The elements of *X* specified by Y

Example:
```
      3 5 ELTOF 1 2 7 8 12 17
7 12
```

Program: *INCREASE* Group: *MANIPULATE*

Syntax: *R←Y INCREASE X* Subroutines: None

Description: Increments selected values of a time series.

Input: *X* = Original time series

Y = Vector of alternating element numbers and increments (or decrements if negative)

Output: *R* = Modified time series

Example:

```
      2 200 4 ¯5 INCREASE 10 20 30 40 50 60
10 220 30 35 50 60
```

Program: *REPLACE* Group: *MANIPULATE*

Syntax: *R←Y REPLACE X* Subroutines: None

Description: Replaces selected values of a time series.

Input: *X* = Original time series
Y = Vector of alternating element numbers and new values

Output: *R* = Modified time series

Example:

```
      2 22 4 44 REPLACE 10 20 30 40 50 60
10 22 30 44 50 60
```

Program: *INIT* Group: *MANIPULATE*

Syntax: *R←Y INIT X* Subroutines: None

Description: Establishes initial conditions.

Input: *X* = Time series
Y = Initial terms for the time series

Output: *R* = *X* with *Y* replacing the initial terms

Example:

```
      10 20 INIT 1 2 3 4 5
10 20 3 4 5
```

Program: *LZF* Group: *MANIPULATE*

Syntax: *R←Y LZF X* Subroutines: None

Description: Generates a time series using left zero fill.

Input: *X* = Time series
Y = Length of resulting time series

Output: *R* = Last *Y* terms of *X*. If *X* is too short, it is padded on the left with zeros.

Example:
```
      5 LZF 10 8
0 0 0 10 8
```

Program: *RZF* Group: *MANIPULATE*

Syntax: *R←Y RZF X* Subroutines: None

Description: Generates a time series using right zero fill.

Input: *X* = Time series
Y = Length of resulting time series

Output: *R* = First *Y* terms of *X*. If *X* is too short, it is padded with zeros on the right.

Example:
```
      5 RZF 10 8
10 8 0 0 0
```

Program: *RLF* Group: *MANIPULATE*

Syntax: *R←Y RLF X* Subroutines: None

Description: Generates a time series using right last fill.

Input: *X* = Time series
Y = Length of resulting time series

Output: *R* = First *Y* terms of *X*. If *X* is too short, it is padded on the right with its last element.

Example:
```
      5 RLF 10 8
10 8 8 8 8
```

Program: *LFF* Group: *MANIPULATE*

Syntax: *R←Y LFF X* Subroutines: None

Description: Generates a time series using left first fill.

Input: *X* = Time series
Y = Length of resulting time series

Output: *R* = Last *Y* terms of *X*. If *X* is too short, it is padded on the left with its first element.

Example:

```
      5 LFF 10 8
10 10 10 10 8
```

Program: *TRANSPOSE* Group: *MANIPULATE*

Syntax: *R←TRANSPOSE X* Subroutines: None

Description: Interchanges the rows and columns of a matrix of data.

Input: *X* = Matrix or vector (treated as a one-row matrix)

Output: *R* = Transposed data

Examples:

```
      X
100 120 150 140 180
150 170 110 210 140
110 190 130 100 130

      TRANSPOSE X
100 150 110
120 170 190
150 110 130
140 210 100
180 140 130

      TRANSPOSE 5 10 20
5
10
20
```

Program: *PLUSMINUS* Group: *MANIPULATE*

Syntax: *R←Y PLUSMINUS X* Subroutines: None

Description: Selects positive or negative terms from a series.

Input: *X* = Time series
Y = 1 or ¯1

Output: *R* = Series modified as follows:

If *Y*=1, negative elements of *Y* are replaced by zeros.

If *Y*=¯1, positive elements of *X* are replaced by zeros, and negative elements are replaced by their absolute values.

Examples:

```
      1 PLUSMINUS 10 ¯10 20 ¯20 30
10 0 20 0 30

      ¯1 PLUSMINUS 10 ¯10 20 ¯20 30
0 10 0 20 0
```

Program: *BCMATX* Group: *MATRIX*

Syntax: *R←Y BCMATX X* Subroutines: None

Description: Builds column matrix.

Input: *X* = Vector or matrix
Y = Vector

Output: *R* = New matrix with *Y* as the next column. The number of columns in *R* is one more than the number of columns in *X*. The number of rows in *R* equals the number of rows in *X*. If *Y* is too short, it is padded with zeros.

Examples:

```
      MAT←4 5 6 BCMATX 1 2 3

      MAT
1 4
2 5
3 6

      7 8 BCMATX MAT
1 4 7
2 5 8
3 6 0
```

Program: *BRMATX* Group: *MATRIX*

Syntax: *R←Y BRMATX X* Subroutines: None

Description: Builds row matrix.

Input: *X* = Vector or matrix
Y = Vector

Output: *R* = New matrix with *Y* as the next row. The number of rows in *R* is one more than the number of rows in *X*. The number of columns in *R* equals the number of columns in *X*. If *Y* is too short, it is padded with zeros.

Examples:

```
      MAT←4 5 6 BRMATX 1 2 3

      MAT
1 2 3
4 5 6

      7 8 BRMATX MAT
1 2 3
4 5 6
7 8 0
```

Program: *MIN* Group: *MINMAX*

Syntax: *R←Y MIN X* Subroutines: None

Description: Extends the minimum primitive function to take the minimum of vectors and matrices.

Input: *X* = Scalar, time series, or matrix
Y = Scalar, time series, or matrix

If neither *X* nor *Y* is a scalar, lengths must conform.

Output: *R* = Y⌊X, performed row by row when operating on a vector and matrix.

Example:

```
      Y
14 16 28 30
18 20 32 35

      Y MIN 20 20 25 30
14 16 25 30
18 20 25 30
```

Program: *MAX* Group: *MINMAX*

Syntax: *R←Y MAX X* Subroutines: None

Description: Extends the maximum primitive function to take the maximum of vectors and matrices.

Input: *X* = Scalar, time series, or matrix
Y = Scalar, time series, or matrix

If neither *X* nor *Y* is a scalar, lengths must conform.

Output: *R* = Y⌈X, performed row by row when operating on a vector and matrix

Example:

```
      Y
14 16 28 30
18 20 32 35

      Y MAX 20 20 30 30
20 20 30 30
20 20 32 35
```

Program: *MINSCAN* Group: *MINMAX*

Syntax: *R←MINSCAN X* Subroutines: None

Description: Minimizes consecutive terms of a time series.

Input: *X* = Time series

Output: *R* = Time series adjusted as follows:
R[1] = *X*[1]
R[*I*] = Minimum of *X*[*I*-1] and *X*[*I*], for *I*>2

Example:
```
      MINSCAN 1 2 3 4 3 2
1 1 2 3 3 2
```

Program: *MAXSCAN* Group: *MINMAX*

Syntax: *R←MAXSCAN X* Subroutines: None

Description: Maximizes consecutive terms of a time series.

Input: *X* = Time series

Output: *R* = Time series adjusted as follows:
R[1] = *X*[1]
R[*I*] = Maximum of *X*[*I*-1] and *X*[*I*], for *I*>2

Example:
```
      MAXSCAN 1 2 3 4 3 2
1 2 3 4 4 3
```

Program: *PDCF* Group: *PTIMEVALUE*

Syntax: *R*←*Y PDCF X* Subroutine: *DCF*

Description: Calculates a probabilistic discounted cash flow.

Input: *X* = Matrix of *K* likely cash streams, one per row

Y[1] = Periodicity and cash flow type
- ±1 ←→ monthly
- ±3 ←→ quarterly
- ±6 ←→ half-yearly
- ±12 ←→ yearly
- \+ ←→ in advance (payment at the beginning of each period)
- \- ←→ in arrears (payment at the end of each period)

Y[2 *TO K*+1] = Probabilities for each cash flow in *X*, summing to 1.

Y[*K*+2 *TO N*] = Nominal annual discount rates, year by year. If enough rates are not provided, the last one is replicated as necessary.

If *Y* consists entirely of probabilities and interest rates, then *Y*←¯1,*Y*.

Output: *R* = Discounted expected (average) cash flow, with the discounting compounded every period.

*Z*1 = 1 + the nominal periodic discount rates (global variable)

Examples:

```
           X
100 110 120 130 140 150
100 100 100 100 100 100
 90  90  80  80  70  70

      3 .3 .45 .25 .08 .1 PDCF X
97.5 98.529 97.078 98.002 96.542 96.891
      Z1
1.02 1.02 1.02 1.02 1.025 1.025

      ¯3 .3 .45 .25 .08 .1 PDCF X
95.588 96.597 95.175 96.08 94.187 94.528
```

*Z*1 is the same as above.

Program: *PFV* Group: *PTIMEVALUE*

Syntax: *R*←*Y* *PFV* *X* Subroutines: *PDCF*, *DCF*

Description: Calculates the probabilistic future value of a collection of cash streams.

Input: *X* = Matrix of *K* likely cash streams, one per row

Y[1] = Periodicity and payment method
- ±1 ↔ monthly
- ±3 ↔ quarterly
- ±6 ↔ half-yearly
- ±12 ↔ yearly
- \+ ↔ in advance (payment at the beginning of each period)
- \- ↔ in arrears (payment at the end of each period)

Y[2 *TO* *K*+1] = Probabilities for each cash flow in *X*, summing to 1.

Y[*K*+2 *TO* *N*] = Nominal annual discount rates, year by year. The last rate entered is replicated so that each element of *X* has a rate.

If *Y* consists entirely of probabilities and interest rates, then *Y*←¯1,*Y*

Output: *R* = Future value of the expected (average) value of *X* compounded at the rates in *Y*. The periodic interest rate is the annual rate in *Y* divided by the number of compoundings per year.

Example:

```
      X
100 110 120 130 140 150
100 100 100 100 100 100
 90  90  80  80  70  70

      3 .3 .45 .25 .08 .1 PFV X
664.7587922
```

Program: *PPV* Group: *PTIMEVALUE*

Syntax: *R*←*Y* *PPV* *X* Subroutines: *PDCF*, *DCF*

Description: Calculates the probabilistic present value for a collection of cash streams.

Input: *X* = Matrix of *K* likely cash streams, one per row

Y[1] = Periodicity and payment method
- ±1 ↔ monthly
- ±3 ↔ quarterly
- ±6 ↔ half-yearly
- ±12 ↔ yearly
- \+ ↔ in advance (payment at the beginning of each period)
- \- ↔ in arrears (payment at the end of each period)

Y[2 *TO* *K*+1] = Probabilities for each cash flow in *X*, summing to 1.

Y[*K*+2 *TO* *N*] = Nominal annual discount rates, year by year. The last rate entered is replicated so that each element of *X* has a rate.

If *Y* consists entirely of probabilities and interest rates, then *Y*←¯1,*Y*

Output: *R* = Present value of the expected (average) value of *X* discounted by the rates in *Y*. The periodic discount rate is the annual rate in *Y* divided by the number of compoundings per year.

Example:

```
      X
100 110 120 130 140 150
100 100 100 100 100 100
 90  90  80  80  70  70

      3 .3 .45 .25 .08 .1 PPV X
584.5419341
```

Program: *DCF* Group: *RATES*

Syntax: *R*←Y *DCF X* Subroutines: None

Description: Calculates a discounted cash flow.

Input: *X* = Cash stream

Y[1] = Periodicity and cash flow type
- ±1 ↔ monthly
- ±3 ↔ quarterly
- ±6 ↔ half-yearly
- ±12 ↔ yearly
- \+ ↔ in advance (payment at the beginning of each period)
- \- ↔ in arrears (payment at the end of each period)

Y[2 *TO N*] = Nominal annual discount rates, year by year. If enough rates are not provided, the last one is replicated as necessary.

If Y consists entirely of probabilities and interest rates, then Y←¯1,Y.

Output: *R* = Discounted cash flow, with the discounting compounded every period.

Z1 = 1 + the nominal periodic discount rates (global variable)

Examples:

```
      3 .08 .1 DCF 100 100 100 100 100 100 100
   100
100 98.039 96.117 94.232 92.385 90.131 87.933
      85.788
      Z1
1.02 1.02 1.02 1.02 1.025 1.025 1.025 1.025

      ¯3 .08 .1 DCF 100 100 100 100 100 100 100
   100
98.039 96.117 94.232 92.385 90.131 87.933 85.788
      83.696
```

Z1 is the same as in the first example.

```
      12 .08 .1 DCF 100 100
100 92.593
      Z1
1.08 1.1

      ¯12 .08 .1 DCF 100 100
92.593 84.175
```

Z1 is the same as in the first example.

Program: *FV* Group: *RATES*

Syntax: *R*←Y *FV X* Subroutine: *DCF*

Description: Calculates the future value of a cash stream.

Input: *X* = Cash stream

Y[1] = Periodicity and payment method
- ±1 ←→ monthly
- ±3 ←→ quarterly
- ±6 ←→ half-yearly
- ±12 ←→ yearly
- + ←→ in advance (payment at the beginning of each period)
- - ←→ in arrears (payment at the end of each period)

Y[2 *TO N*] = Nominal annual discount rates, year by year. The last rate entered will be replicated so that each element of *X* has a rate.

If Y consists entirely of probabilities and interest rates, then Y←¯1,Y.

Output: *R* = Future value of *X* compounded at the rates in Y. The periodic interest rate is the annual rate in Y divided by the number of compoundings per year.

Example:

```
      3 .06 FV 1000 2000 3000 4000 5000
15532.94623
```

Program: *PV* Group: *RATES*

Syntax: *R*←*Y* *PV* *X* Subroutine: *DCF*

Description: Calculates the present value of a cash stream.

Input: *X* = Cash stream

Y[1] = Periodicity and payment method
- ±1 ←→ monthly
- ±3 ←→ quarterly
- ±6 ←→ half-yearly
- ±12 ←→ yearly
- \+ ←→ in advance (payment at the beginning of each period)
- \- ←→ in arrears (payment at the end of each period)

Y[2 *TO* *N*] = Nominal annual discount rates, year by year. The last rate entered is replicated so that each element of *X* has a rate.

If *Y* consists entirely of probabilities and interest rates, then *Y*←¯1,*Y*.

Output: *R* = Present value of *X* discounted by the rates in *Y*. The periodic discount rate is the annual rate in *Y* divided by the number of compoundings per year.

Example:

```
      3 .06 PV 1000 2000 3000 4000 5000
14418.61772
```

Program: *ROR*

Group: *RATES*

Syntax: *R*←Y *ROR X*

Subroutines: *IROR*, *DIROR*, *SFROR*, *AROR*, *SAROR*, *MSAROR*, *IDF* and their subroutines as selected: *DCF*, *PV*, *FV*, *SFCFL*

Description: Calculates selectable rate of return.

Input: *X* = Cash stream, including both revenues and expenses

Y[1] = Rate of return
- 1 ↔ internal rate of return
- 2 ↔ discounted rate of return
- 3 ↔ sinking fund rate of return
- 4 ↔ annuity rate of return
- 5 ↔ savings account rate of return
- 6 ↔ modified savings account rate of return

Y[2] = Periodicity
- 1 ↔ monthly
- 3 ↔ quarterly
- 12 ↔ yearly

Y[3 *TO N*] = Any remaining input parameters for each rate of return type

Output: *R* = Rate of return

Examples: For descriptions of the input parameters and examples of the computations, see the documentation for the programs *IROR*, *DIROR*, *SFROR*, *AROR*, *SAROR*, and *MSAROR*.

Program: *IROR* Group: *RATES*

Syntax: *R*←Y *IROR X* Subroutine: *IDF*

Description: Calculates an internal rate of return.

Input: *X* = Cash stream, including both revenues and expenses

Y[1] = Model periodicity
1 ↔ monthly
3 ↔ quarterly
6 ↔ half-yearly
12 ↔ yearly

Y[2] = Optional -- Type of rate when periodicity is not yearly
0 ↔ produce the effective annual rate (default)
1 ↔ produce the nominal annual rate

Output: *R* = The annual internal rate of return of the cash stream. No reinvestment of positive stream benefits is assumed; the result is the rate needed to yield a net present value of zero. For multiple rates of return, the rate closest to 10% is calculated.

Example:
```
      12 IROR ¯300 150 100 200 50 ¯100 50 100
.265032032
```

Program: *MIROR* Group: *RATES*

Syntax: *R*←Y *MIROR X* Subroutine: *MIDF*

Description: Calculates several internal rates of return.

Input: *X* = Matrix of cash streams, including both revenues and expenses

Y[1] = Model periodicity
- 1 ←→ monthly
- 3 ←→ quarterly
- 6 ←→ half-yearly
- 12 ←→ yearly

Y[2] = Optional -- Type of rate when periodicity is not yearly
- 0 ←→ produce the effective annual rates (default)
- 1 ←→ produce the nominal annual rates

Output: *R* = Vector of annual internal rates of return of the cash streams. No reinvestment of positive stream benefits is assumed; the result is the rates needed to yield net present values of zero. For multiple rates of return for an individual stream, the rate closest to 10% is calculated.

Example:

```
      FLOW
¯300 ¯150  100  200   50 ¯100   50  100
¯300 ¯100   50   50  100  100  150  200

      12 MIROR FLOW
.265032032 .1032940015
```

Program: *AROR* Group: *RATES*

Syntax: *R←Y AROR X* Subroutines: *DCF*, *IDF*, *IROR*, *PV*

Description: Calculates an annuity rate of return.

Input: *X* = Cash stream, including both revenues and expenses

Y[1] = Model periodicity
- 1 ←→ monthly
- 3 ←→ quarterly
- 6 ←→ half-yearly
- 12 ←→ yearly

Y[2 *TO N*] = Annual annuity discount rates for each period

Output: *R* = The annuity rate of return of the cash stream; that is, the program *IROR* is applied to the cash flow based on the present value of all equity investments and negative flows at the user-supplied discount rates. The last rate entered is replicated for any unspecified period. For multiple annuity rates of return, the rate closest to 10% is calculated.

Example:

```
      12 .05 AROR ¯300 150 100 200 50 ¯100 50 100
.2007165422
```

Program: *DIROR* Group: *RATES*

Syntax: *R←Y DIROR X* Subroutines: *DCF*, *IDF*, *IROR*

Description: Calculates a discounted internal rate of return.

Input: *X* = Cash stream, including both revenues and expenses

Y[1] = Model periodicity
- 1 ←→ monthly
- 3 ←→ quarterly
- 6 ←→ half-yearly
- 12 ←→ yearly

Y[2 *TO N*] = Annual discount rates for each period

Output: *R* = The discounted internal rate of return of the cash stream; that is, the program *IROR* is applied to the cash flow discounted at the annual rates supplied. The last rate entered is replicated for any unspecified period. For multiple discounted rates of return, the rate closest to 10% is calculated.

Example:

```
      12 .05 DIROR ¯300 150 100 200 50 ¯100 50 100
.2047924115
```

Program: *MSAROR* Group: *RATES*

Syntax: *R*←Y *MSAROR X* Subroutines: *DCF*, *FV*, *IDF*, *IROR* *PV*, *SAROR*, *SFCFL*

Description: Calculates a modified savings account rate of return.

Input: *X* = Cash stream, including both revenues and expenses

Y[1] = Model periodicity
- 1 ←→ monthly
- 3 ←→ quarterly
- 6 ←→ half-yearly
- 12 ←→ yearly

Y[2 *TO N*] = Annual sinking fund and discount rates for each period

Output: *R* = Modified savings account rate of return of the cash stream; that is, the program *IROR* is applied to the cash flow modified by the sinking fund method and then by the savings account method. See *SFROR* and *SAROR*. The last rate entered is replicated for any unspecified period. For multiple modified savings account rates of return, the rate closest to 10% is calculated.

Example:

```
      12 .05 MSAROR ¯300 150 100 200 50 ¯100 50
   100
.112806275
```

Program: *SAROR* Group: *RATES*

Syntax: *R*←*Y* *SAROR* *X* Subroutines: *DCF*, *FV*, *IDF*, *IROR*, *PV*

Description: Calculates a savings account rate of return.

Input: *X* = Cash stream, including both revenues and expenses

Y[1] = Model periodicity
- 1 ←→ monthly
- 3 ←→ quarterly
- 6 ←→ half-yearly
- 12 ←→ yearly

Y[2 *TO* *N*] = Annual savings account discount rates for each period

Output: *R* = The savings account rate of return of the cash stream; that is, the program *IROR* is applied to the cash flow based on the present value of all equity investments and negative flows at the user-supplied discount rates. The rates are subject to any remaining positive benefits being compounded forward at the given reinvestment rates to the period of the final stream element. The last rate entered is replicated for any unspecified period. For multiple savings account rates of return, the rate closest to 10% is calculated.

Example:

```
      12 .05 SAROR ¯300 150 100 200 50 ¯100 50 100
.1017196382
```

Program: *SFROR* Group: *RATES*

Syntax: *R←Y SFROR X* Subroutines: *DCF, IDF, IROR, SFCFL*

Description: Calculates a sinking fund rate of return.

Input: *X* = Cash stream, including both revenues and expenses

Y[1] = Model periodicity
 1 ↔ monthly
 3 ↔ quarterly
 6 ↔ half-yearly
 12 ↔ yearly

Y[2 *TO N*] = Annual sinking fund rates for each period

Output: *R* = The sinking fund rate of return of the cash stream; that is, the program *IROR* is applied to the cash flow based on the assumption of reinvestment of sufficient positive stream benefits at the user-supplied rates to fund all future negative flows. The last rate entered is replicated for any unspecified period. For multiple sinking fund rates of return, the rate closest to 10% is calculated.

Example:
```
      12 .05 SFROR ¯300 150 100 200 50 ¯100 50 100
.247249389
```

Program: *DCIROR* Group: *RATES*

Syntax: *R←Y DCIROR X* Subroutine: *DCIDF*

Description: Calculates an internal rate of return for a combination of discrete and continuous cash flows.

Input: *X* = Two-row matrix with one more column than the number of periods represented by the cash streams. The first row, *D←X*[1;], consists of the discrete cash stream, including both revenues and expenses. *D*[1] is the flow at the beginning of the first period (end of the zeroth period) and *D*[*I*] is the flow at the end of period *I*-1, for *I* = 2,3,...,*N*+1. The second row, *C←X*[2;], consists of the continuous flows, each of which is spread uniformly within each period. *C*[1] must be 0, and *C*[*I*] equals the flow received during period *I*-1, for *I* = 2,3,...,*N*+1.

Y[1] = Model periodicity
- 1 ↔ monthly
- 3 ↔ quarterly
- 6 ↔ half-yearly
- 12 ↔ yearly

Y[2] = Optional (except if *Y*[3]≠0) -- Type of rate when periodicity is not yearly
- 0 ↔ produce effective annual rate (default)
- 1 ↔ produce the nominal annual rate

Y[3] = Optional -- Compounding (discounting) type
- 0 ↔ compound *D* periodically and *C* continuously (default)
- 1 ↔ compound both *D* and *C* continuously
- 2 ↔ compound both *D* and *C* periodically
- 3 ↔ compound *D* continuously and *C* periodically

Output: *R* = Annual internal rate of return based on the combination of *D* and *C*. No reinvestment of positive stream benefits is assumed. For multiple rates of return, the rate closest to 10% is calculated.

```
Examples:          X
           ¯300 ¯150  100  200   50 ¯100   50  100
              0 ¯100  ¯50    0   50  100  150  200

                12 DCIROR X
           .264621608

                12 0 1 DCIROR X
           .2485793479

                12 0 2 DCIROR X
           .2820552504

                12 0 3 DCIROR X
           .2630637962
```

Program: *IDF* Group: *RATES*

Syntax: *R*←Y *IDF X* Subroutines: None

Description: Calculates an internal discount factor.

Input: *X* = Cash stream, including both revenues and expenses
Y = Initial approximation of the internal discount factor

Output: *R* = The internal discount factor of the cash stream. No reinvestment of positive stream benefits is assumed; the result is the discount factor needed to yield a net present value of zero. For multiple factors, the rate closest to .9 is calculated.

Discount factor = 1÷(1+rate of return).

```
Example:        .9 IDF ¯300 150 100 200 50 ¯100 50 100
          .7904938173
```

Program: *MIDF*

Group: *RATES*

Syntax: *R*←Y *MIDF X*

Subroutines: None

Description: Calculates several internal discount factors.

Input: *X* = Matrix of cash streams, including both revenues and expenses

Y = Initial approximation(s) of the internal discount factors

Output: *R* = Vector of internal discount factors of the cash streams. No reinvestment of positive stream benefits is assumed; the result is the discount factors needed to yield net present values of zero.

Discount factor = 1÷(1+rate of return).

Example:

```
      FLOW
¯300 ¯150  100  200   50 ¯100   50  100
¯300 ¯100   50   50  100  100  150  200

      .9 MIDF FLOW
.7904938173 .9063767216
```

Program: *DCIDF* Group: *RATES*

Syntax: *R*←*Y DCIDF X* Subroutines: None

Description: Calculates an internal discount factor for a combination of discrete and continuous cash flows.

Input: *X* = Two-row matrix with *N*+1 columns for *N* periods represented by the cash streams. The first row, *D*←*X*[1;], consists of the discrete cash stream, including both revenues and expenses. *D*[1] equals the flow at the beginning of the first period (end of the zeroth period) and *D*[*I*] equals the flow at the end of period *I*-1, for *I* = 2,3,...,*N*+1. The second row, *C*←*X*[2;], consists of the continuous flows, each spread uniformly within each period. *C*[1] must be 0, and *C*[*I*] equals the flow received during period *I*-1, for *I* = 2,3,...,*N*+1.

Y[1] = Initial approximation of the internal discount factor

Y[2] = Optional -- Compounding (discounting) type
0 ←→ compound *D* periodically and *C* continuously (default)
1 ←→ compound both *D* and *C* continuously
2 ←→ compound both *D* and *C* periodically
3 ←→ compound *D* continuously and *C* periodically

Output: *R* = The internal discount factor based on the combination of *D* and *C*. No reinvestment of positive stream benefits is assumed.

Discount factor = 1÷(1+rate of return).

Examples:

```
       X
¯300 ¯150  100  200   50 ¯100   50  100
   0 ¯100  ¯50    0   50  100  150  200

      .9 DCIDF X
.7907503665

      .9 1 DCIDF X
.8009102518

      .9 2 DCIDF X
.7799975856

      .9 3 DCIDF X
.791725646
```

Program: *SFCFL* Group: *RATES*

Syntax: *R←Y SFCFL X* Subroutine: *DCF*

Description: Calculates a sinking fund cash flow.

Input: *X* = Original cash flow

Y[1] = Periodicity and cash flow type
±1 ←→ monthly
±3 ←→ quarterly
±6 ←→ half-yearly
±12 ←→ yearly
\+ ←→ in advance (payment at the beginning of each period)
\- ←→ in arrears (payment at the end of each period)

Y[2 *TO N*] = Nominal annual interest rates, year by year. If enough rates are not provided, the last one will be replicated as necessary.

If *Y* consists entirely of interest rates, then *Y*←¯1,*Y*.

Output: *R* = Modified cash flow where early positive flows are set aside into a sinking fund to fund all future negative flows. This fund pays interest at the rates specified in *Y*. After all negative flows are funded, the result consists of the remainder of the positive flows. Only flows from the first positive one to the last negative one are affected. All others will remain unchanged. The sinking fund will be considered infeasible if the positive flows cannot cover subsequent negative flows. In this case, *R*=*X*.

Examples:

```
      12 0 SFCFL ¯300 100 250 ¯200 50 100 ¯90
¯300 100 50 0 50 10 0

      12 0 SFCFL ¯300 100 250 ¯200 50 100 ¯90
   ¯100
¯300 100 10 0 0 0 0 0

      12 0 SFCFL ¯300 100 50 ¯200 250 100 ¯90
*** INCOMING CASH FLOWS INSUFFICIENT TO COVER
    FUTURE_OUTFLOWS
¯300 100 50 ¯200 250 100 ¯90

      12 .1 SFCFL ¯300 100 250 ¯200 50 100 ¯90
¯300 100 68.182 0 50 18.182 0

      12 .1 SFCFL ¯300 100 250 ¯200 50 100 ¯90
   ¯100
¯300 100 61.072 0 0 0 0 0
```

```
Program:  INCOMEASSETS                                  Group:  RATIOS

Syntax:  R←Y INCOMEASSETS X                        Subroutine: DIV


Description:  Calculates the ratio of income statement items
              to balance sheet items.


Input:  X = Balance sheet time series

        Y[1] = Periodicity
                1 ←→ monthly
                3 ←→ quarterly
               12 ←→ yearly

        Y[2 TO N] = Income statement time series.  The
                    length of Y must be one more than the
                    length of X, and X and Y must have
                    enough data to account for a whole
                    number of years.

Output:  R = Ratio of each income item by the period average
             of the balance sheet items.

Example:  In this example, X represents quarterly balances
          for two years.  The yearly average balances are
          120 and 200.

                X
          100 110 125 145 170 200 210 220

                3 24 30 45 36 44 50 65 80 INCOMEASSETS X
          .2 .25 .375 .3 .22 .25 .325 .4

------------------------------------------------------------

Program:  PER                                           Group:  RATIOS

Syntax:  R←Y PER X                                  Subroutine:  DIV


Description:  Calculates percentages.


Input:  X = Time series used as the divisor
        Y = Time series used as the dividend

Output:  R = Y percent of X, or 100×(Y÷X)

Example:          2 3 4 5 6 PER 5 6 8 5 4
          40 50 50 100 150
```

Program: *RND* Group: *ROUNDING*

Syntax: *R←Y RND X* Subroutines: None

Description: Rounds to a given number of decimal places.

Input: *X* = Time series
Y = Number of decimal places for rounding

Output: *R* = Rounded time series

Examples:

```
      1 RND 23.7452 23.75 23.7548
23.7 23.8 23.8

      2 RND 23.7452 23.75 23.7548
23.75 23.75 23.75
```

Program: *ROUND* Group: *ROUNDING*

Syntax: *R←Y ROUND X* Subroutines: None

Description: Rounds data to some multiple of a given unit.

Input: *X* = Numeric vector or matrix

Y[1] = Unit amount used for rounding

Y[2] = Optional -- Type of rounding (default = 0)
0 ←→ round to nearest unit
1 ←→ round up to next unit
¯1 ←→ round down to next unit

Output: *R* = Rounded data

Examples:

```
      X
1242.3 1888.7 2110.9 2504.8 2636.1 2774.5

      50 ROUND X
1250 1900 2100 2500 2650 2750

      50 1 ROUND X
1250 1900 2150 2550 2650 2800

      50 ¯1 ROUND X
1200 1850 2100 2500 2600 2750

      .5 ROUND X
1242.5 1888.5 2111 2505 2636 2774.5
```

Program: *ROUNDUP* Group: *ROUNDING*

Syntax: *R←Y ROUNDUP X* Subroutines: None

Description: Rounds an array of data up to a desired power of 10.

Input: *X* = Data array (scalar, vector, or matrix)

Y = Desired power(s) of 10. *Y* can also be an array as long as its length conforms with the length of *X*.

Output: *R* = Rounded data

Examples:

```
      2 ROUNDUP 3456 4563 5634
3500 4600 5700

      1 2 3 ROUNDUP 3456 4563 5634
3460 4600 6000

      1 2 3 ROUNDUP 3456
3460 3500 4000
```

Program: *ROUNDDOWN* Group: *ROUNDING*

Syntax: *R←Y ROUNDDOWN X* Subroutines: None

Description: Rounds an array of data down to a desired power of 10.

Input: *X* = Data array (scalar, vector, or matrix)

Y = Desired power(s) of 10. *Y* can also be an array as long as its length conforms with the length of *X*.

Output: *R* = Rounded data

Examples:

```
      2 ROUNDDOWN 3456 4563 5634
3400 4500 5600

      1 2 3 ROUNDDOWN 3456 4563 5634
3450 4500 5000

      1 2 3 ROUNDDOWN 3456
3450 3400 3000
```

Program: *FEDTAX79* Group: *TAXES*

Syntax: *R*←Y *FEDTAX79 X* Subroutine: *MQY* (only if Y[1]≠12)

Description: Calculates federal tax, based on the 1979 rate structure.

Input: *X* = Vector representing a profit or loss stream

Y[1] = Data periodicity
1 ↔ monthly
3 ↔ quarterly
6 ↔ half-yearly
12 ↔ yearly

Y[2] = Optional (if Y[3] omitted or 0) -- Tax loss carry forward (0 or negative)

Y[3] = Optional -- Starting year of data in *X* (0 ↔ 1979 or after)

A pre-1979 starting year may be supplied so that some or all of the taxes may be calculated according to the old rate structure.

Output: *R* = Taxes payable
Z1 = Tax loss carry forward (global variable)

The lengths of both *R* and Z1 match the length of *X*.

Examples:

```
      12 FEDTAX79 20000 30000 ¯10000 60000
3400 5250 0 9250
      Z1
0 0 ¯10000 0

      12 ¯40000 FEDTAX79 20000 30000 ¯10000 60000
0 1700 0 9250
      Z1
¯20000 0 ¯10000 0

      12 0 1978 FEDTAX79 20000 30000 ¯10000 60000
4400 5250 0 9250

      3 FEDTAX79 20000 30000 ¯10000 60000
5350 8025 ¯2675 16050
      Z1
0 0 0 0

      3 FEDTAX79 20000 30000 ¯10000 ¯60000
0 0 0 0
      Z1
0 0 0 ¯20000
```

Notes: The U.S. corporate tax law now provides that income starting in 1979 be taxed at the following rates:

- 17% on the first $25000
- 20% on the second $25000
- 30% on the third $25000
- 40% on the fourth $25000
- 46% on any excess over $100,000

The old rate structure for pre-1979 income was

- 22% on the first $25000
- 48% on any excess over $25000

Because of the sliding tax scale, the income stream *X* should be in absolute dollar amounts.

If the income stream is not yearly, the program annualizes it, calculates the annual tax burden, and distributes the tax into the periods proportional to the income in each period. Any tax loss carry forward is placed in the last period of the year.

The program can handle a fractional number of years (if periodicity is not yearly). In this case, any tax loss carry forward for the partial year is lost.

While *FEDTAX*79 contains the correct tax rates, it does not completely simulate the complexities of the U.S. tax code because it simplifies the sophisticated carry back/carry forward rules into an indefinite carry forward only. The program should therefore be used as a modeling tool and not as a self-contained tax accounting system.

Program: *ITC* Group: *TAXES*

Syntax: *R←ITC X* Subroutines: None

Description: Calculates investment tax.

Input: *X*[1] = Purchase price
X[2] = Stated life in months
X[3] = Actual life in months

Output: *R*[1] = Initial investment tax credit
R[2] = Refund, if any

A 10% investment tax credit is assumed.

Example:

```
      ITC 1000000 60 48
66666.67 ¯33333.33
```

Program: *CUMRANGE* Group: *TESTS*

Syntax: *R←Y CUMRANGE X* Subroutines: None

Description: Maintains a cash balance within an approximate range.

Input: *X* = Vector of cash flows over successive periods

Y[1] = Rounding factor (the number of powers of 10)
Y[2] = Minimum of the range
Y[3] = Maximum of the range
Y[4] = Opening cash balance

Output: *R* = Vector of cash balances produced by the flow kept approximately within the specified range. Any excess cash over the maximum is rounded down to even 10*Y[1]'s and any cash shortage under the minimum is rounded up to even 10*Y[1]'s. The result has the same length as *X*.

Notes: The expression:

```
3 25000 40000 10000 CUMRANGE X
```

produces a result with an opening balance of 10000 and, depending on *X*, a closing balance between 25000 and 40000 (possibly a little more). Any excess cash is rounded down to even thousands and any cash shortage is rounded up to the next thousand.

Example:

```
      3 25000 40000 10000 CUMRANGE 13300 22300
   30000 ¯50000
 25300 40600 40600 25600
```

Program: *CUMZEROMAX* Group: *TESTS*

Syntax: *R←Y CUMZEROMAX X* Subroutines: None

Description: Calculates a time series of balances from an opening balance and a cash flow, as long as the resulting balances are not less than zero.

Input: *X* = Cash flow time series
Y = Opening balance

Output: *R* = Non-negative time series of closing balances

Examples:

```
          F
 ¯5 10 ¯15 12 ¯10 4

          0 CUMZEROMAX F
 0 10 0 12 2 6

          8 CUMZEROMAX F
 3 13 0 12 2 6

          18 CUMZEROMAX F
 13 23 8 20 10 14
```

Program: *INTRANGE* Group: *TESTS*

Syntax: *R←Y INTRANGE X* Subroutines: None

Description: Tests for an integer vector within a range.

Input: *X* = Numeric vector

Y[1] = Lower limit
Y[2] = Upper limit

Output: *R* = Boolean vector; 1s correspond to the positions in *X* that are integers within the specified range.

Example:

```
          3 7 INTRANGE 1 1.5 2 2.5 3 3.5 4 4.5 5
 0 0 0 0 1 0 1 0 1
```

Program: *TRANGE* Group: *TESTS*

Syntax: *R←Y TRANGE X* Subroutines: None

Description: Tests to see if data is within a certain range.

Input: *X* = Time series

Y[1] = Low limit
Y[2] = High limit

Output: *R* = Boolean vector; 1s indicate that the value is within the specified range

Example:

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

Program: *RANGE* Group: *TESTS*

Syntax: *R←Y RANGE X* Subroutines: None

Description: Limits data to certain range.

Input: *X* = Time series

Y[1] = Low limit
Y[2] = High limit

Output: *R* = Time series *X* filtered between the limits

Example:

```
      2 6 RANGE 1 2 3 4 5 6 7
2 2 3 4 5 6 6
```

Program: *MAXHALT* Group: *TESTS*

Syntax: *R←Y MAXHALT X* Subroutines: None

Description: Halts a program if a maximum is exceeded.

Input: *X* = Input time series
Y = Allowable maximum value

Output: *R* = *X* if maximum is not exceeded, or a vector of zeros if maximum is exceeded by any element of *X*. Prior to returning such result, the program halts; that is, the program *MAXHALT* is suspended.

Example: *COST ← 1E6 MAXHALT UNITS×PRICE*

Program: *MINHALT* Group: *TESTS*

Syntax: *R←Y MINHALT X* Subroutines: None

Description: Halts a program if a minimum is exceeded.

Input: *X* = Input time series
Y = Allowable minimum value

Output: *R* = *X* if minimum is not exceeded, or a vector of zeros if minimum is exceeded by any element of *X*. Prior to returning such result, the program halts; that is, the program *MINHALT* is suspended.

Example: *REVENUE ← 0 MINHALT UNITS×PRICE*

Program: *ANNUITY* Group: *TIMEVALUE*

Syntax: *R←ANNUITY X* Subroutines: None

Description: Calculates various uniform annuity results.

Input: *X*[1] = Type of calculation
- ¯1 ←→ calculate the present value of an annuity
- 0 ←→ calculate the time series of periodic values of the annuity
- 1 ←→ calculate the ultimate value of the annuity

X[2] = Periodicity
- 1 ←→ monthly
- 3 ←→ quarterly
- 6 ←→ half-yearly
- 12 ←→ yearly

X[3] = Periodic payment

X[4] = Number of periods

X[5] = Annual interest rate

X[6] = Optional -- Type of payment (default = 0)
- 0 ←→ in arrears (payment at the end of each period)
- 1 ←→ in advance (payment at the beginning of each period)

Output: *R* = Annuity calculation specified by *X*[1]

Examples: The following examples calculate ultimate value.

```
      ANNUITY 1 1 50 120 .05
7764.113972

      ANNUITY 1 1 50 120 .05 1
7796.464447
```

The following examples calculate successive balances.

```
      ANNUITY 0 6 100 5 .1
100 205 315.25 431.0125 552.563125

      ANNUITY 0 6 100 5 .1 1
105 215.25 331.0125 452.563125 580.1912812
```

The following examples calculate present value.

```
      ANNUITY ¯1 1 50 60 .06
2586.278038

      ANNUITY ¯1 1 50 60 .06 1
2599.209428
```

CHAPTER 3

THE *FORECAST* WORKSPACE

The *FORECAST* workspace contains programs that:

- parametrically generate new data
- conversely, derive appropriate parameters from empirical data
- combine these approaches by deriving parameters from history and using them to project the future.

The workspace also contains programs to perform regressions, to seasonally adjust data, and to smooth results.

CYCLE 3-2
DERIVETREND 3-4
EXPOSMOOTH 3-10
EXPOTREND 3-13
FORETREND 3-15
RELATION 3-18
SMOOTHING 3-20
TREND 3-22

Program: *DSEASFACTOR* Group: *CYCLE*

Syntax: *R←Y DSEASFACTOR X* Subroutines: *DIV*, *SPREAD*

Description: Derives seasonal adjustment factors for a time series.

Input: *X* = Time series (covering at least two years; can be a fractional number of years)

Y = Periodicity
1 ↔ monthly
3 ↔ quarterly
6 ↔ half-yearly

Output: *R* = Multiplicative periodic adjustment factors (averaging 1; length of *R* equals 12÷*Y*)

Z1 = Seasonally adjusted time series (equals *X* ÷ (ρ*X*)ρ*R*) (global variable)

Example:

```
      3 4ρX        Reshape 3 years of quarterly data
45  76  46  54
22  65  68  68
36  39  52  84

      3 DSEASFACTOR X
0.614 1.049 1.096 1.242

      3 4ρZ1
73.35  72.48  41.98  43.47
35.86  61.99  62.06  54.74
58.68  37.19  47.46  67.62
```

Notes: The seasonal adjustment algorithm is based on a multiple regression technique. For a more intricate seasonal adjustment technique, the *X11* method (not included) should be investigated.

Program: *SEASFACTOR* Group: *CYCLE*

Syntax: *R←Y SEASFACTOR X* Subroutines: None

Description: Seasonalizes a time series.

Input: *X* = Time series
Y = Seasonal or cyclical factors

Output: *R* = Time series scaled by seasonal or cyclical factors. The length of the cycle is determined by the length of *Y*. With monthly factors, each month is multiplied by that month's factor, regardless of year. With quarterly factors, each quarter is multiplied by that quarter's factor, irrespective of year.

Examples:

```
      X
10 10 10 10 10 10 10 10 10 10 10 10 20 20 20 20
      20 20 20 20 20 20 20 20

      FAC
1.1 .8 1.1 1 1.1 1 1.2 1.3 1 1.4 .9 1.2

      FAC SEASFACTOR X
11 8 11 10 11 10 12 13 10 14 9 12 22 16 22 20 22
      20 24 26 20 28 18 24

      1.1 .8 1.2 .9 SEASFACTOR 10 10 10 10 20 20
    20 20
11 8 12 9 22 16 24 18
```

Program: *DTTREND1* Group: *DERIVETREND*

Syntax: *R←DTTREND1 X* Subroutines: *REG, DIV, DIV0*

Description: Derives first order (linear) time trend coefficients.

Input: *X* = Time series

Output: *R*[1] = *N* = number of terms in the time series
R[2] = *A* = intercept
R[3] = *B* = slope
R[4] = Standard error of *B*
R[5] = Computed t-value
R[6] = Standard error of estimate
R[7] = Simple correlation coefficient *R*
R[8] = *R**2

The time series *X* is fit by the least squares method to the line *A* + *B*×*T* for *T*=1,2,3,...,*N*.

Example:

```
      DTTREND1 10 12 15 20 28 40 57
7 ¯4 7.5 1.180 6.355 6.245 .943 .890

      T←1 TO 7 ◇ ¯4+7.5×T
3.5 11 18.5 26 33.5 41 48.5
```

Program: *DTTREND2*   Group: *DERIVETREND*

Syntax: *R*←*DTTREND2 X*   Subroutines: *REG*, *DIV*, *DIVO*

Description: Derives second order (quadratic) time trend coefficients.

Input: *X* = Time series

Output: *R*[1] = *N* = number of terms in the time series
R[2] = *A* = intercept
R[3] = *B* = linear coefficient
R[4] = *C* = quadratic coefficient
R[5] = Standard error of *B*
R[6] = Computed t-value for *B*
R[7] = Standard error of *C*
R[8] = Computed t-value for *C*
R[9] = Standard error of estimate
R[10] = Simple correlation coefficient *R*
R[11] = *R*⋆2

The time series *X* is fit by the least squares method to the parabola *A* + (*B*×*T*) + *C*×*T*⋆2 for *T*=1,2,3,...,*N*.

Example:

```
      DTTREND2 10 12 15 20 28 40 57
7 14 ¯4.5 1.5 1.094 4.114 .134 11.225 1.225 .998
      .997

      T←1 TO 7 ◇ 14+T×¯4.5+T×1.5
11 11 14 20 29 41 56
```

Program: *DSTEPTREND* Group: *DERIVETREND*

Syntax: *R*←*Y* *DSTEPTREND* *X* Subroutines: *DTTREND*1, *REG*
DIV, *DIV*0

Description: Derives the parameters of a step trend function.

Input: *X* = Time series of length *N*.

Y = Partitioning numbers for *X* consisting of positive whole numbers whose sum is *N*. The combination of the number of elements in each line segment is used to approximate the original time series.

Output: *R* = Triplets of parameters that can be used as input to *STEPTREND* to approximate *X* by a series of line segments. Each triplet consists of the starting value, the slope of the line segment, and the number of elements comprising the line segment.

Example:

```
      TS←(1 TO 10)*2 ◇ TS
1 4 9 16 25 36 49 64 81 100

      PAR←2 4 4 DSTEPTREND TS ◇ PAR
1 3 2 8 9 4 48 17 4

      STEPTREND PAR
1 4 8 17 26 35 48 65 82 99
```

Program: *DPOWER* Group: *DERIVETREND*

Syntax: *R*←Y *DPOWER X* Subroutines: None

Description: Derives *N*th order (power series) time trend coefficients.

Input: *X* = Time series
Y = The power *N* of the power series. Y must be less than the length of *X*.

Output: *R*[1] = Y
R[2] = Length of *X*
R[3] = Constant term *C*0
R[4] = Linear coefficient *C*1
.
.
.
R[3+Y] = Coefficient *C*Y of *T*⋆Y

The time series *X* is fit by the least squares method to the power series (polynomial):

*C*0 + (*C*1×*T*) + ... + (*C*Y×*T*⋆Y)

for *T* = 1 to *R*[2]

Examples:

```
      2 DPOWER 10 12 15 20 28 40 57
 2 7 14 ¯4.5 1.5

      3 DPOWER 10 12 15 20 28 40 57
 3 7 8 2.33333 ¯0.5 0.16667
```

Program: *DGROWTH* Group: *DERIVETREND*

Syntax: *R←Y DGROWTH X* Subroutine: *AGR*

Description: Derives growth rate parameters.

Input: *X* = Time series

```
Y[1] = Periodicity
        1 ←→ monthly
        3 ←→ quarterly
        6 ←→ half-yearly
       12 ←→ yearly

Y[2] = Optional -- Parameter selection switch
       0 ←→ derive a single pair of parameters (A,G)
            from all of the data in X (default)
       1 ←→ derive pairs of parameters (A,G) for
            each year's data in X.  Available only
            if periodicity is not yearly.
```

Output: *R* = *N,A,G* if *Y*[2] = 0 or omitted
N,A1,G1,A2,G2,... if *Y*[2] = 1 and *Y*[1] ≠ 12

The data in *X*, or each year's data if *Y*[2] = 1, is fit by least squares to the model:

X = *A*×(1+*G*÷*FREQ*)⋆1 *TO K*

where *A* = base value, *G*÷*FREQ* is the periodic growth rate, *FREQ* = 12÷*Y*[1], and *K* = number of terms in the segment of *X* that is fit to the model.
N = *R*[1] is the length of the time series *X*. The growth rate *G* (or *Gi*) in *R* is an annualized nominal growth rate. If any element of *X* is non-positive, the corresponding growth rate is 0.

Examples: In the following examples, the program derives growth parameters for eight years, two years, and for each of two years, respectively.

```
      X←10 11 13 16 20 25 31 37

      12 DGROWTH X
8 7.591 0.2174

      3 DGROWTH X
8 7.591 0.8696

      3 1 DGROWTH X
8 8.292 0.6833 16.44 0.9153
```


Program: *AGR* Group: *DERIVETREND*

Syntax: *R←Y AGR X* Subroutines: None

Description: Fits a time series to a growth curve.

Input: *X* = Time series

Y = Periodicity
1 ←→ monthly
3 ←→ quarterly
6 ←→ half-yearly
12 ←→ yearly

Output: *R* = Two-element vector. The time series *X* is fit by least squares to the model:

X = *A*×(1+*G*)⋆1 *TO N*

where *N* = length of *X*, *A* = base value, and *G* = periodic growth rate.

R[1] = *A*, the base value

R[2] = Annualized nominal growth rate (*G*×frequency). *R*[2] equals zero if any element of *X* is non-positive.

Examples:

```
      X
10 11 13 16 20 25 31 37

      ⎕←R←12 AGR X
7.591 0.2174

      R[1]×(1+R[2])⋆1 TO 8
9.241 11.25 13.7 16.67 20.3 24.71 30.08 36.62

      3 AGR X
7.591 0.8696
```

Program: *EXPOSMOOTH1* Group: *EXPOSMOOTH*

Syntax: *R←Y EXPOSMOOTH1 X* Subroutines: None

Description: Performs single exponential smoothing on a time series when the following model is appropriate: $X[T] = A + \epsilon[T]$ where $\epsilon[T]$ is random noise, for $T = 1,2,3,\ldots$.

Input: *X* = Historical time series to be smoothed

Y[1] = Smoothing constant α, $0<\alpha<1$
Y[2] = Number of periods to be forecast after *X*
Y[3] = Initial estimate of the constant level *A*

Output: *R* = Forecast for the series of length *Y*[2]
Z1 = Original historical series *X* (global variable)
Z2 = Smoothed historical series (global variable)

Examples:

```
          X
14 11 12 9 11 7 6 9 12 10 14

          .2 5 10 EXPOSMOOTH1 X   Long term smoothing
10.6 10.6 10.6 10.6 10.6          (Less fluctuation)
          Z2
10.8 10.8 11.1 10.7 10.7 9.98 9.18 9.15 9.72 9.77
          10.6

          .2 5 0 EXPOSMOOTH1 X    (More sensitive to
9.76 9.76 9.76 9.76 9.76          initial estimate)
          Z2
2.8 4.44 5.95 6.56 7.45 7.36 7.09 7.47 8.38 8.7
          9.76

          .8 5 10 EXPOSMOOTH1 X   Short term smoothing
13.3 13.3 13.3 13.3 13.3          (Wider fluctuation)
          Z2
13.2 11.4 11.9 9.58 10.7 7.74 6.35 8.47 11.3 10.3
          13.3

          .8 5 0 EXPOSMOOTH1 X    (Less sensitive to
13.3 13.3 13.3 13.3 13.3          initial estimate)
          Z2
11.2 11 11.8 9.56 10.7 7.74 6.35 8.47 11.3 10.3
          13.3
```

Program: *EXPOSMOOTH2* Group: *EXPOSMOOTH*

Syntax: *R*←Y *EXPOSMOOTH2 X* Subroutines: None

Description: Performs double exponential smoothing on a time series when the following model is appropriate: $X[T] = A + (B \times T) + \epsilon[T]$ where $\epsilon[T]$ is random noise, for $T = 1,2,3,\ldots$.

Input: *X* = Historical time series to be smoothed

Y[1] = Smoothing constant α, $0<\alpha<1$
Y[2] = Number of periods to be forecast after *X*
Y[3] = Initial estimate of the constant level *A*
Y[4] = Initial estimate of the trend factor *B*

Output: *R* = Forecast for the series of length Y[2].

Z1 = Original historical series *X* (global variable)

Z2 = Smoothed historical series, adjusted for trend (global variable)

Z3 = Sequence of adaptively calculated estimates for *B* (global variable)

Example:

```
      X
15 13 15 13 17 16 22 19 28 25

      .2 5 10 2 EXPOSMOOTH2 X
27.6 29.3 31.1 32.8 34.5
      Z2
13.1 14.4 15.9 16.1 17.6 18.1 20.6 21.1 24.6 25.9
      Z3
2.12 2.03 1.97 1.78 1.74 1.61 1.7 1.57 1.78 1.73
```

Program: *EXPOSMOOTH3* Group: *EXPOSMOOTH*

Syntax: *R←Y EXPOSMOOTH3 X* Subroutines: None

Description: Performs triple exponential smoothing on a time series when the following model is appropriate: $X[T] = A + (B\times T) + (.5\times C\times T*2) + \epsilon[T]$ where $\epsilon[T]$ is random noise, for $T = 1,2,3, \ldots$

Input: *X* = Historical time series to be smoothed

Y[1] = Smoothing constant α, 0<α<1
Y[2] = Number of periods to be forecast after *X*
Y[3] = Initial estimate of the constant level *A*
Y[4] = Initial estimate of the trend factor *B*
Y[5] = Initial estimate of the quadratic factor *C*

Output: *R* = Forecast for the series of length *Y*[2].

Z1 = Original historical series *X* (global variable)

Z2 = Smoothed historical series, quadratically adjusted (global variable)

Z3 = Sequence of adaptively calculated estimates for *B* (global variable)

Z4 = Sequence of adaptively calculated estimates for *C* (global variable)

Example:

```
      X
15 13 15 13 17 16 22 19 28 25

      .2 5 10 2 .4 EXPOSMOOTH3 X
30.2 33.5 36.9 40.6 44.5
      Z2
13.6 14.8 16.4 16.3 18 18.5 21.5 21.8 26.1 27.2
      Z3
2.7 2.75 2.85 2.53 2.63 2.42 2.79 2.48 3.11 2.89
      Z4
0.422 0.395 0.373 0.321 0.305 0.267 0.275 0.232
      0.26 0.225
```

Program: *EXPOGROWTH* Group: *EXPOTREND*

Syntax: *R←EXPOGROWTH X* Subroutines: None

Description: Forecasts based on exponential growth.

Input: *X*[1] = Number *N* of terms to forecast
X[2] = *K* = base amount (initial value)
X[3] = *B* = coefficient of time

Output: *R* = Exponential growth curve generated by the equation *K*×⋆*B*×*T* where ⋆ = ⋆1, for *T* = 0,1,2,...,*N*-1

Example:

```
      EXPOGROWTH 8 1000 .1
1000 1105.17 1221.4 1349.86 1491.82 1648.72
      1822.12 2013.75
```

Program: *ASYMPTOTE* Group: *EXPOTREND*

Syntax: *R←ASYMPTOTE X* Subroutines: None

Description: Forecasts asymptotically reaching upper limit.

Input: *X*[1] = Number *N* of periods to forecast
X[2] = *B* = coefficient of time
X[3] = *K* = upper limit (asymptotic value)

Output: *R* = Time series generated from *K*×(1-⋆-*B*×*T*) where ⋆ = ⋆1, for *T* = 0,1,2,...,*N*-1

Example:

```
      ASYMPTOTE 8 .4 1000
0 329.68 550.67 698.81 798.1 864.66 909.28 939.19
```

Program: *SCURVE* Group: *EXPOTREND*

Syntax: *R←SCURVE X* Subroutines: None

Description: Generates values along a saturation (logistic) curve.

Input: *X* = Vector of parameters:

- *X*[1] = Initial time period
- *X*[2] = Initial value (>0)
- *X*[3] = Intermediate time period (>*X*[1])
- *X*[4] = Intermediate value (>*X*[2])
- *X*[5] = Ending time period for the output (>*X*[3])
- *X*[6] = Either the ultimate saturation value (>*X*[4], if *X*[7] is 0 or omitted) or the time period in which the rate of increase is greatest and the value is half-way to saturation (if *X*[7]≠0)
- *X*[7] = Parameter type for *X*[6] (default = 0 if *X*[7] absent)

Output: *R* = Time series of values along the *SCURVE* at each time period from *X*[1] to *X*[5], inclusive. If *X*[5] is less than *X*[1], the values will be projected backwards in time from *X*[1] to *X*[5].

Examples:

```
      SCURVE 2 100 5 200 12 500
100 128.72 162.33 200 240.19 280.89 320 355.71
      386.84 412.9 433.99

      SCURVE 2 100 5 200 12 6 1
100 128.9 162.56 200 239.53 279.07 316.5 350.17
      379.06 402.89 421.89
```

Notes: In the first example, the value is 433.99 in period 12, while the ultimate saturation value is 500.

In the second example, the values are half-way to saturation at 239.53 in period 6. If more terms (*X*[5]) were taken, the values along the curve would approach 479.06.

The program assumes that the values will asymptotically approach zero as time proceeds backwards.

Program: *FTTREND1* Group: *FORETREND*

Syntax: *R*←Y *FTTREND1 X* Subroutines: *DTTREND1*, *REG*, *DIV*, *DIVO*

Description: Forecasts based on a first order (linear) time trend.

Input: *X* = Historical time series

Y[1] = Number of periods to forecast

Y[2] = Optional -- Number of periods of history to include in the output (default = 0).

Output: *R* = Time series with the last Y[2] terms of *X*, followed by forecasts for Y[1] periods into the future, using the linear model *A+B×TIME*.

Example:

```
      5 3 FTTREND1 10 12 15 20 28 40 57
28 40 57 56 63.5 71 78.5 86
```

Program: *FTTREND2* Group: *FORETREND*

Syntax: *R*←Y *FTTREND2 X* Subroutines: *DTTREND2*, *REG*, *DIV*, *DIVO*

Description: Forecasts based on a second order (quadratic) time trend.

Input: *X* = Historical time series

Y[1] = Number of periods to forecast

Y[2] = Optional -- Number of periods of history to include in the output (default = 0).

Output: *R* = Time series with the last Y[2] terms of *X* followed by forecasts for Y[1] periods into the future, using the quadratic model *A+(B×TIME)+C×TIME*2*.

Example:

```
      5 3 FTTREND2 10 12 15 20 28 40 57
28 40 57 74 95 119 146 176
```

Program: *FPOWER* Group: *FORETREND*

Syntax: *R←Y FPOWER X* Subroutine: *DPOWER*

Description: Forecasts based on an *N*th order (power series) time trend.

Input: *X* = Historical time series

Y[1] = Power *N* of the power series, less than the length of *X*

Y[2] = Number of periods to forecast

Y[3] = Optional -- Number of periods of history to include in the output (default = 0).

Output: *R* = Time series with the last Y[3] terms of *X* followed by forecasts for Y[2] periods into the future. The time series is generated by the power series (polynomial):

$$C0 + (C1 \times T) + \ldots + (CN \times T \star N)$$

where *N* = Y[1] and time *T* corresponds to the respective elements of *X*.

Example:

```
      3 7 4 FPOWER 10 12 15 20 28 40 57
20 28 40 57 80 110 148 195 252 320 400
```

Program: *FGROWTH* Group: *FORETREND*

Syntax: *R*←Y *FGROWTH X* Subroutine: *AGR*

Description: Forecasts based on a growth model.

Input: *X* = Historical time series

Y[1] = Number of periods to forecast

Y[2] = Optional -- Number of periods of prior history to include (default = 0)

Output: *R* = Time series with the last Y[2] periods of history followed by Y[1] periods of forecast. *X* is fit by least squares to the model *X* = *A*×(1+*G*)*1 *TO N* where *A* = base value, *G* = periodic growth rate, and *N* = number of terms in *X*. *A* and *G* are used to extend the time series for Y[1] more periods. If any element of *X* is non-positive, the growth rate is assumed to be 0.

Example:

```
      8 4 FGROWTH 10 11 13 16 20 25 31 37
20 25 31 37 44.58 54.28 66.08 80.44 97.93 119.2
      145.1 176.7
```

Program: *FDIFF* Group: *RELATION*

Syntax: *R*←Y *FDIFF X* Subroutines: None

Description: Forecasts based on the average historical difference of two time series.

Input: *X* = Independent time series, includes both history and forecast

Y = Dependent time series, includes only history

Output: *R* = Time series consisting of the history *Y* followed by a forecast derived by adding the average of the historical differences *Y-X* to the forecast data in *X*.

Examples:

```
      1 6 6 16 FDIFF 1 2 3 4 5 6
1 6 6 16 9.75 10.75

      1 3 2 4 FDIFF 1 2 3 4 5 6
1 3 2 4 5 6
```

Program: *FREGRESS* Group: *RELATION*

Syntax: *R*←Y *FREGRESS X* Subroutines: None

Description: Forecasts based on a regression of two time series.

Input: *X* = Independent time series, includes both history and forecast

Y = Dependent time series, includes only history

Output: *R* = Time series consisting of the history *Y* followed by a forecast based on the forecast data in *X*. A regression *Y* = *A* + *B*×*X* uses the historical data. The parameters *A* and *B* are used to derive the forecast for *Y*.

Example:

```
      1 6 6 16 FREGRESS 1 2 3 4 5 6
1 6 6 16 18.5 23
```

Program: *FRATIO* Group: *RELATION*

Syntax: *R←Y FRATIO X* Subroutine: *DIV*

Description: Forecasts based on the average historical ratio of two time series.

Input: *X* = Independent time series, includes both history and forecast

Y = Dependent time series, includes only history

Output: *R* = Time series consisting of the history *Y* followed by a forecast derived by multiplying the forecast data in *X* by the average of the historical ratios *Y÷X*.

Example:

```
      1 6 6 16 FRATIO 1 2 3 4 5 6
1 6 6 16 12.5 15
```

Program: *FTRENDRATIO* Group: *RELATION*

Syntax: *R←Y FTRENDRATIO X* Subroutines: *DIV*, *DIVO* *DTTREND1*, *REG*

Description: Forecasts based on the first order (linear) time trend of the historical ratio of two time series.

Input: *X* = Independent time series, includes both history and forecast

Y = Dependent time series, includes only history

Output: *R* = Time series consisting of the history *Y* followed by a forecast derived by multiplying the forecast data in *X* by projected ratios based on a linear time trend of the historical ratios *Y÷X*.

Example:

```
      1 6 6 16 FTRENDRATIO 1 2 3 4 5 6
1 6 6 16 22.5 31.8
```

Program: *MOVINGAVE* Group: *SMOOTHING*

Syntax: *R←Y MOVINGAVE X* Subroutines: None

Description: Calculates the moving average of a time series.

Input: *X* = Time series

Y[1] = Number of periods to be averaged

Y[2] = Optional -- Average type
 1 ←→ leading edge average
 0 ←→ midpoint average (default)
 ¯1 ←→ trailing edge average

If Y[1] is even and Y[2] equals zero, then Y[1] is raised to the next higher odd integer.

Output: *R* = Smoothed time series. The extreme values of *X* are extended to produce the averages at the endpoints.

Examples:

```
      3 MOVINGAVE 3 9 6 12 9 18 15 3 12
5 6 9 9 13 14 12 10 9

      3 1 MOVINGAVE 3 9 6 12 9 18 15 3 12
3 5 6 9 9 13 14 12 10

      3 ¯1 MOVINGAVE 3 9 6 12 9 18 15 3 12
6 9 9 13 14 12 10 9 12
```

Program: *MOVMAXSCAN* Group: *SMOOTHING*

Syntax: *R←Y MOVMAXSCAN X* Subroutines: None

Description: Calculates a moving maximum with variable effect width.

Input: *X* = Time series
Y = Number of periods for smoothing

Output: *R* = Smoothed time series whose *I*th term equals the maximum of terms *X*[*I*] through *X*[*I*+Y-1]

Example:

```
      3 MOVMAXSCAN 1 7 2 6 5 8 3
7 7 6 8 8 8 3
```

Program: *MOVMINSCAN* Group: *SMOOTHING*

Syntax: *R*←Y *MOVMINSCAN X* Subroutines: None

Description: Calculates a moving minimum with variable effect width.

Input: *X* = Time series
Y = Number of periods for smoothing

Output: *R* = Smoothed time series whose *I*th term equals the minimum of terms *X*[*I*] through *X*[*I*+Y-1]

Example:

```
      3 MOVMINSCAN 1 7 2 6 5 8 3
1 2 2 5 3 3 3
```

Program: *WEIGHTEDAVE* Group: *SMOOTHING*

Syntax: *R*←Y *WEIGHTEDAVE X* Subroutines: None

Description: Calculates the weighted moving average of a time series.

Input: *X* = Time series

Y[1] = *N* = number of periods to be averaged

Y[2 *TO* *N*+1] = Weights for each period

Y[*N*+2] = Average type (optional)
1 ←→ leading edge average
0 ←→ midpoint average (default)
¯1 ←→ trailing edge average

If Y[1] is even and Y[2] equals zero, then Y[2] is changed to 1.

Output: *R* = Smoothed time series. The extreme values of *X* are extended to produce the averages at the endpoints.

Examples:

```
      3 .3 .5 .2 WEIGHTEDAVE 3 9 6 12 9 18 15 3
   12
4.2 6.6 8.1 9.6 11.7 14.7 13.5 8.4 9.3

      3 .3 .5 .2 1 WEIGHTEDAVE 3 9 6 12 9 18 15 3
   12
3 4.2 6.6 8.1 9.6 11.7 14.7 13.5 8.4

      3 .3 .5 .2 ¯1 WEIGHTEDAVE 3 9 6 12 9 18 15
   3 12
6.6 8.1 9.6 11.7 14.7 13.5 8.4 9.3 12
```

Program: *TTREND1* Group: *TREND*

Syntax: *R←TTREND1 X* Subroutines: None

Description: Forecasts using a first order (linear) time trend.

Input: *X*[1] = Number of periods to forecast
X[2] = *A* = intercept of the line
X[3] = *B* = slope of the line

Output: *R* = *A* + *B*×*T* for *T*=1 to *X*[1]

Example:
```
      TTREND1 5 100 10
110 120 130 140 150
```

Program: *TTREND2* Group: *TREND*

Syntax: *R←TTREND2 X* Subroutines: None

Description: Forecasts using a second order (quadratic) time trend.

Input: *X*[1] = Number of periods to forecast
X[2] = *A* = intercept of the parabola
X[3] = *B* = linear coefficient
X[4] = *C* = quadratic coefficient

Output: *R* = *A* +(*B*×*T*) + *C*×*T*⋆2 for *T*=1 to *X*[1]

Example:
```
      TTREND2 5 100 10 .1
110.1 120.4 130.9 141.6 152.5
```

Program: *STEPTREND* Group: *TREND*

Syntax: *R←STEPTREND X* Subroutines: None

Description: Forecasts using a step trend function.

Input: *X* = Succession of triplets, one triplet for each linear segment of the function.

For *I* = 0,1,2,..., each triplet consists of:

X[1+3×*I*] = Height at the left end of the line segment
X[2+3×*I*] = Slope of the line segment
X[3+3×*I*] = Number of terms in the segment

Output: *R* = Time series consisting of the successive line segments

Example:

```
      STEPTREND 5 1 3 8 .75 4 11 .5 6
5 6 7 8 8.75 9.5 10.25 11 11.5 12 12.5 13 13.5
```

Program: *POWER* Group: *TREND*

Syntax: *R←POWER X* Subroutines: None

Description: Forecasts using an *N*th order (power series) time trend.

Input: *X*[1] = Number of periods to forecast
X[2] = Constant term
X[3] = Linear coefficient
X[4] = Quadratic coefficient
.
.
.
X[*N*+2] = Coefficient of *T*⋆*N*

Output: *R* = *X*[2] + (*X*[3]×*T*) + (*X*[4]×*T*⋆2) + ... + *X*[*N*+2]×*T*⋆*N* for *T*=1 to *X*[1]

Example:

```
      POWER 5 100 10 .1 .01
110.11 120.48 131.17 142.24 153.75
```

Program: *GROWTH* Group: *TREND*

Syntax: *R←GROWTH X* Subroutines: None

Description: Forecasts using various growth rates.

Input: *X*[1] = Number of periods to be forecast

X[2] = Periodicity
  1 ←→ monthly
  3 ←→ quarterly
  6 ←→ half-yearly
  12 ←→ yearly

X[3] = Base value

X[4 *TO N*] = Annual growth rates for each period. If necessary, the last rate entered will be replicated to provide a rate for each period.

Output: *R* = Time series generated by growing the base value by the given annual rates, compounded each period.

Examples:

```
      GROWTH 4 12 100 .05 .1
105 115.5 127.05 139.75

      GROWTH 8 3 100 .05 .05 .05 .05 .1
101.25 102.52 103.8 105.09 107.72 110.41
      113.18 116
```

Program: *DORG* Group: *TREND*

Syntax: *R←Y DORG X* Subroutines: None

Description: Calculates dollar amount or growth rate.

Input: *X* = Time series of dollar amounts (greater than or equal to 1) or growth rates (less than 1).

Y = Opening dollar amount

Output: *R* = Time series of dollar amounts. The opening balance *Y* is grown at the rates given in *X*. However, when terms of *X* are larger than 1, they are inserted in *R* as actual dollar amounts.

Example:

```
      100 DORG .2 .1 150 .1 200 .1 .05
120 132 150 165 200 220 231
```

CHAPTER 4

THE *PROBABILITY* WORKSPACE

The *PROBABILITY* workspace contains routines to generate random sample according to various probability distributions. These distributions are classified as either analytic (when the distribution function can be expressed as a mathematical formula) or empirical (when the user supplies the shape of the distribution function). Each of the routines generates multiple samples. For the analytic distribution generators, any parameters required may vary from sample to sample.

EMPIRICAL 4-2
ANALYTIC 4-7
DISTFN 4-42

Program: *CUMDISCRETE* Group: *EMPIRICAL*

Syntax: *R←N CUMDISCRETE C* Subroutine: *NUNIFORM*

Description: Returns samples from the discrete distribution whose cumulative probability function is specified by *C*.

Input: *N* = Number of samples desired

C[1] = First *X* value, *X*1
C[2] = Cumulative probability at *X*1 (probability of *X*1)
C[3] = Second value, *X*2
C[4] = Cumulative probability at *X*2
.
.
.
C[¯1+2×*K*] = *K*th value
C[2×*K*] = Cumulative probability at the *K*th value

Output: *R* = *N* samples from the discrete distribution with cumulative probability function *C*.

Notes: *CUMDISCRETE* requires that *C*[2] be positive, that {*C*[odd]} be increasing, and that {*C*[even]} be increasing. Ideally, *C*[2×*K*] should be 1, but the program scales the cumulative probabilities to satisfy this condition.

See the documentation for *DISCRETE* for an alternate way to generate samples from the discrete distribution.

Program: *DISCRETE* Group: *EMPIRICAL*

Syntax: *R←N DISCRETE D* Subroutine: *NUNIFORM*

Description: Returns samples from the discrete distribution specified by *D*.

Input: *N* = Number of samples desired

D[1] = First *X* value, *X*1
D[2] = Probability of *X*1
D[3] = Second value, *X*2
D[4] = Probability of *X*2
.
.
.
D[¯1+2×*K*] = *K*th value
D[2×*K*] = Probability of the *K*th value

Output: *R* = *N* samples from the discrete distribution specified by *D*.

Notes: *DISCRETE* requires that {*D*[odd]} contain no duplicates, and that {*D*[even]} be non-negative and not all zero. If the sum of the "probability" values is not 1, the program scales them to satisfy this condition.

Program: *HISTOGRAM* Group: *EMPIRICAL*

Syntax: *R←N HISTOGRAM D* Subroutine: *NUNIFORM*

Description: Produces random samples from a histogram distribution.

Input: *N* = Number of samples desired

D = Numeric vector specifying the histogram to be used as the probability density function as follows:

D[1] = Left endpoint of first interval
D[2] = 0
D[3] = Right endpoint of first interval
D[4] = Height of first interval
D[5] = Right endpoint of second interval
D[6] = Height of second interval
.
.
.
D[1+2×*K*] = Right endpoint of *K*th interval
D[2+2×*K*] = Height of *K*th interval

Output: *R* = *N* samples from histogram specified by *D*.

Notes: The endpoints *D*[1 3 5 ...] should be strictly increasing. The heights specified should be non-negative and not all 0. Ideally, the area under the histogram should be 1, but the program automatically adjusts the heights to satisfy this condition.

Program: *POLYGON* Group: *EMPIRICAL*

Syntax: *R←N POLYGON D* Subroutine: *NUNIFORM*

Description: Produces random samples from a polygon distribution.

Input: *N* = Number of samples desired

D = Numeric vector specifying the polygon to be used as the probability density function as follows:

D[1] = First *X* value
D[2] = First height
D[3] = Second *X* value
D[4] = Second height
.
.
.
D[¯1+2×*K*] = *K*th *X* value
D[2×*K*] = *K*th height

Output: *R* = *N* samples from *D* polygon distribution.

Notes: The *X* values *D*[1 3 5 ...] should be strictly increasing. The heights specified should be non-negative and not all 0. Ideally, the area under the polygon should be 1, but the program automatically adjusts the heights to satisfy this condition.

Program: *POLYGONCUM* Group: *EMPIRICAL*

Syntax: *R←N POLYGONCUM C* Subroutine: *NUNIFORM*

Description: Returns samples from a histogram distribution whose cumulative distribution function is the polygon specified by *C*.

Input: *N* = Number of samples desired

C[1] = Left endpoint of the first interval
C[2] = 0
C[3] = Right endpoint of the first interval
C[4] = Cumulative probability at *C*[3]
C[5] = Right endpoint of the second interval
C[6] = Cumulative probability at *C*[5]
.
.
.
C[1+2×*K*] = Right endpoint of *K*th interval
C[2+2×*K*] = Cumulative probability at *C*[1+2×*K*]

Output: *R* = *N* samples from the histogram distribution whose cumulative distribution function is polygon *C*.

Notes: *POLYGONCUM* requires that {*C*[odd]} be increasing and {*C*[even]} be increasing. Ideally, *C*[2+2×*K*] should be 1, but the program automatically adjusts the heights to satisfy this condition.

The program *CUMHISTOGRAM* is equivalent to *POLYGONCUM* since if a polygon (collection of straight line segments) is used as a cumulative distribution function, the underlying density function must be a histogram (step function). See the documentation for *HISTOGRAM* for an alternate method of generating samples from a histogram distribution.

Program: *CUMHISTOGRAM* Group: *EMPIRICAL*

Syntax: *R←N CUMHISTOGRAM C* Subroutines: *POLYGONCUM* *NUNIFORM*

Description: Equivalent to (and calls) the program *POLYGONCUM* when generating samples from a histogram distribution with a known cumulative distribution function. See the documentation for *POLYGONCUM* for complete details.

Program: *PBETA* Group: *ANALYTIC*

Syntax: *R←P PBETA Q* Subroutines: *NUNIFORM, PGAMMA, ΔPOWER*

Description: Returns samples from the beta distribution(s).

Input: *P* = Power parameter, a scalar or vector
Q = Power parameter, a scalar or vector

If *P* and *Q* are both vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample corresponds to its respective distribution parameters. The length of *R* depends on the length of the arguments.

Notes: The density function for the beta distribution with parameters (*P*,*Q*) is

```
F(X) = ((G(P+Q)÷G(P)×G(Q))×(X*P-1)×(1-X)*Q-1
```

for 0<*X*<1 and *P*,*Q*>0, where *G*(∘) is the gamma function. Statistics for this distribution are

```
MEAN = P÷P+Q
VARIANCE = (P×Q)÷(P+Q+1)×(P+Q)*2
MODE = (P-1)÷P+Q-2
RANGE = [0,1]
```

The important beta distribution is bell-shaped if *P*,*Q*>1 and skewed right or left depending on whether *P*-*Q* is positive or negative. The greater *P* and *Q* are, the greater the kurtosis (spikey-ness) is. When *P*=*Q*=1, the distribution is uniform, and when *P*,*Q*<1 the distribution is convex (higher at the endpoints).

The beta distribution can be generalized by the transformation

```
Z←LOW+(HIGH-LOW)×P PBETA Q
```

and the random variable *Z* has

```
MEAN = LOW+(HIGH-LOW)×P÷P+Q
VARIANCE = VARIANCE ABOVE×(HIGH-LOW)*2
MODE = LOW+(HIGH-LOW)×(P-1)÷P+Q÷2
RANGE = [LOW,HIGH]
```

Program: *PBINOMIAL* Group: *ANALYTIC*

Syntax: *R←N PBINOMIAL P* Subroutine: *NUNIFORM*

Description: Returns samples from the binomial distribution(s).

Input: *N* = Scalar or vector representing the number of independent trials

P = Scalar or vector representing the probability of success in each trial

Output: *R* = Samples from the distribution. An individual sample corresponds to its respective parameters in *N* and *P*. The length of *R* depends on the lengths of *N* and *P*.

Notes: The binomial distribution is a discrete distribution which represents the number *X* of successes in *N* independent trials, where *P* is the probability of success on each trial. The probability function is

```
F(X) = (X!N)×(P*X)×(1-P)*N-X
```

where *X* = 0,1,2,...,*N*.

Selected statistics are

```
MEAN = N×P
VARIANCE = N×P×1-P
MODE = N⌊⌊P×N+1 as well as
       N⌊(P×N+1)-1 if P×N+1 is an integer
SKEW = (1-2×P)÷(N×P×1-P)*.5
KURTOSIS = 3+(1-6×P×1-P)÷N×P×1-P
RANGE = {0,1,2,...,N}
```

Program: *PCAUCHY* Group: *ANALYTIC*

Syntax: *R←M PCAUCHY W* Subroutines: *NCAUCHY*, *NUNIFORM*

Description: Returns samples from the Cauchy distribution(s).

Input: *M* = Scalar or vector representing the mean parameter(s)
W = Scalar or vector representing the width parameter(s)

If *M* and *W* are both vectors, their lengths must conform.

Output: *R* = Samples from the Cauchy distribution. An individual sample in *R* corresponds to its respective distribution parameters. The length of *R* depends on the lengths of *M* and *W*.

Notes: The density function for the Cauchy distribution with parameters (*M*,*W*) is

```
F(X) = 1÷(○W)×1+((X-M)÷W)*2
```

Selected statistics are

```
MEAN = M
VARIANCE = INFINITY
MEDIAN = M
MODE = M
SKEW = 0
RANGE = (-INFINITY,INFINITY)
```

Program: *NCAUCHY* Group: *ANALYTIC*

Syntax: *R←NCAUCHY N* Subroutine: *NUNIFORM*

Description: Returns *N* samples from the "standard" Cauchy distribution with density function

```
F(X) = 1÷(○1)×1+X*2
```

Input: *N* = Scalar or vector representing the number of samples to be returned from the distribution

Output: *R* = *N* samples corresponding from the distribution. The length of *R* equals *N*.

Notes: This distribution can be generalized by the transformation:

```
Z←M+W×NCAUCHY N
```

and the random variable *Z* has

```
MEAN = M
VARIANCE = INFINITY
MEDIAN = M
MODE = M
SKEW = 0
RANGE = (-INFINITY,INFINITY)
```

See the documentation for *PCAUCHY*.

Program: *PCHISQUARED* Group: *ANALYTIC*

Syntax: *R←PCHISQUARED DF* Subroutines: *NUNIFORM*, *PBETA* *PGAMMA*, *∆POWER*

Description: Returns samples from the (central) chi-square distribution.

Input: *DF* = Scalar or vector representing degrees of freedom to use in the distribution(s)

Output: *R* = Samples from the distribution. An individual sample in *R* is chosen from the chi-square distribution with the corresponding degrees of freedom in *DF*. The length of *R* is the same as the length of *DF*.

Notes: The density function is

```
F(X) = (÷(2*DF÷2)×G(DF÷2))×(X*(DF÷2)-1)×*-X÷2
```

for *X*>0, where *G*(•) is the gamma function. Selected statistics are

```
MEAN = DF
VARIANCE = 2×DF
MODE = 0⌈DF-2
SKEW = (2*1.5)÷DF*.5
KURTOSIS = 3+12÷DF
RANGE = (0,INFINITY)
```

A chi-square distribution with *DF* degrees of freedom is equivalent to 2×*PGAMMA DF*÷2. *PCHISQUARED* 2 is the same as *PEXPONENTIAL* 2. See the documentation for *PEXPONENTIAL* and *PGAMMA* for details.

Program: *PNONCENCHISQ* Group: *ANALYTIC*

Syntax: *R←NC PNONCENCHISQ DF* Subroutines: *NNORMAL*, *NUNIFORM*, *PBETA*, *PCHISQUARED*, *PGAMMA*, *∆POWER*

Description: Returns samples from the non-central chi-square distribution(s).

Input: *NC* = Scalar or vector non-centrality parameter(s)
DF = Scalar or vector representing degrees of freedom

If both *NC* and *DF* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to the respective elements in *NC* and *DF*. The length of *R* depends on the lengths of *NC* and *DF*.

Notes: A non-central chi-square (*NC*,*DF*) distribution is the sum of the squares of *DF* independent normal distributions with means *MU*[*I*] (1≤*I*≤*DF*) and variance 1. The parameter *NC* is defined as (+/*MU*⋆2)⋆.5.

Selected statistics are

```
MEAN = DF+(NC⋆2)
VARIANCE = (2×DF)+(4×NC⋆2)
```

Some authors use a non-centrality parameter defined as *NC*1←(+/*MU*⋆2)÷2. To convert, use the relationship *NC*←(2×*NC*1)⋆.5.

Program: *PEXPONENTIAL* Group: *ANALYTIC*

Syntax: *R←PEXPONENTIAL M* Subroutine: *NUNIFORM*

Description: Returns samples from the exponential distribution(s).

Input: *M* = Scalar or vector representing the mean

Output: *R* = Samples from the distribution. An individual sample in *R* comes from the exponential distribution with corresponding mean in *M*. The length of *R* is the length of *M*.

Notes: The density function of the exponential distribution is

```
F(X) = (1÷M)×*-X÷M for X≥0
```

Selected statistics are

```
MEAN = M
VARIANCE = M*2
MODE = 0
MEDIAN = M×⍟2
SKEW = 2
KURTOSIS = 9
RANGE = (0,INFINITY)
```

Program: *NEXPONENTIAL* Group: *ANALYTIC*

Syntax: *R←NEXPONENTIAL N* Subroutine: *NUNIFORM*

Description: Returns samples from the "standard" exponential distribution.

Input: *N* = Scalar or vector representing the number of samples

Output: *R* = The desired samples from the distribution

Notes: The distribution has the density function

*F(X) = *-X* for *X≥0*

This basic distribution can be generalized by the transformation

Z←LOW+WIDTH×NEXPONENTIAL N

Selected statistics for *Z* are

```
MEAN = LOW+WIDTH
VARIANCE = WIDTH*2
MODE = LOW
MEDIAN = LOW+WIDTH×⍟2
SKEW = 2
KURTOSIS = 9
RANGE = (0,INFINITY)
```

See also the documentation for *PEXPONENTIAL*.

Program: *NEXTREME* Group: *ANALYTIC*

Syntax: *R←NEXTREME N* Subroutine: *NUNIFORM*

Description: Returns *N* samples from the standard Type-I extreme distribution.

Input: *N* = Number of samples to generate from the distribution.

Output: *R* = Samples from the distribution.

Notes: The density function for this distribution is

F(X) = *-X+*-X (for all *X*)

and the cumulative distribution function is

PROB(RANDOM VARIABLE ≤ *X)* = *-*-X (for all *X*).

Program: *PF* Group: *ANALYTIC*

Syntax: *R←M PF N* Subroutines: *NUNIFORM*, *PBETA*, *PGAMMA*, *∆POWER*

Description: Returns samples from the F-distribution(s).

Input: *M* = Scalar or vector representing degrees of freedom
N = Scalar or vector representing degrees of freedom

If *M* and *N* are both vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to the respective degrees of freedom in *M* and *N*. The length of *R* depends on the length of *M* and *N*.

Notes: The density function for the F-distribution with *M*,*N* degrees of freedom (for numerator and denominator, respectively) is

$$F(X) = \frac{G(.5\times M+N)\times(M\star M\div 2)\times(N\star N\div 2)\times X\star(M\div 2)-1}{G(M\div 2)\times G(N\div 2)\times(N+M\times X)\star.5\times M+N}$$

where $X\geq 0$ and $G(\bullet)$ is the gamma function. Selected statistics are

MEAN = *N*÷*N*-2 if *N*>2
VARIANCE = (2×(*N*⋆2)×*M*+*N*-2)÷*M*×(*N*-4)×(*N*-2)⋆2 if *N*>4
MODE = (*N*×*M*-2)÷*M*×*N*+2
RANGE = [0,*INFINITY*)

Program: *PNON* with *PCENTRALF* Group: *ANALYTIC*

Syntax: *R←NC PNON DF1 PCENTRALF DF2* Subroutines: *NNORMAL NUNIFORM, PBETA PCHISQUARED, PGAMMA PNONCENCHISQ, ΔPOWER*

Description: The combination of the programs *PNON* and *PCENTRALF* returns samples from the non-central F-distribution(s).

Input: *NC* = Non-centrality parameter(s)
DF1 and *DF2* = Degrees of freedom

The lengths of parameters *NC*, *DF1*, and *DF2* must conform: all scalars, all vectors, or a combination as long as the lengths of vectors longer than one match.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to the respective elements of *NC*, *DF1*, and *DF2*. The length of *R* depends on the lengths of *NC*, *DF1*, and *DF2*.

Notes: This distribution is defined as the ratio of a non-central chi-square distribution with non-centrality parameter *NC* and degrees of freedom *DF1* and a (central) chi-square distribution with *DF2* degrees of freedom.

Refer to the documentation for the programs *PNONCENCHISQ* and *PCHISQUARED* for more information about the distribution parameters.

Program: *PCENTRALF* Group: *ANALYTIC*

Refer to the documentation for the program *PNON* for information about how to use *PNON* in combination with *PCENTRALF* to generate samples from non-central F-distributions.

Program: *PGAMMA* Group: *ANALYTIC*

Syntax: *R← PGAMMA P* Subroutines: *NUNIFORM*, *PBETA*, *∆POWER*

Description: Returns samples from the gamma distribution(s).

Input: *P* = Scalar or vector of power parameter(s)

Output: *R* = Samples from the distribution. An individual sample in *R* is chosen from the gamma distribution with the corresponding parameter in *P*. The length of *R* is the length of *P*.

Notes: The density function of the gamma distribution is

```
F(X) = (1÷G(P))×(X*P-1)×*-X
```

for 0≤*X* and 0<*P*, where *G*(•) is the gamma function.

This basic gamma distribution can be generalized by the transformation

```
Z←LOW+WIDTH×PGAMMA P
```

and the random variable *Z* has

```
MEAN = LOW+WIDTH×P
VARIANCE = P×WIDTH*2
MODE = LOW+WIDTH×0⌈P-1
SKEW = 2×P*¯.5
KURTOSIS = 3+6÷P
RANGE = [LOW,INFINITY)
```

The Erlang distribution is the same as the gamma distribution, but usually an Erlang-type *K* distribution refers to a gamma distribution where

```
P = Positive integer K
LOW = 0
WIDTH = 1÷K×LAMBDA
```

so that

```
MEAN = 1÷LAMBDA
VARIANCE = 1÷K×LAMBDA*2
```

Program: *PGEOMETRIC* Group: *ANALYTIC*

Syntax: *R←PGEOMETRIC P* Subroutine: *NUNIFORM*

Description: Returns samples from the geometric distribution(s) with parameter(s) *P*.

Input: *P* = Probability of success on each independent trial.

Output: *R* = Samples from the geometric distribution. An individual sample in *R* comes from the geometric distribution with corresponding parameter in *P*. The length of *R* is the length of *P*.

Notes: The geometric distribution is a discrete distribution which represents the number *X* of failures before the first success in a sequence of independent trials, where *P* is the probability of success on each trial. The probability function is

```
F(X) = P×(1-P)*X where X = 0,1,2,...
```

Selected statistics are

```
MEAN = (1-P)÷P
VARIANCE = (1-P)÷P*2
MEDIAN = ⌈¯1+(1-P)⍟.5
MODE = 0
SKEW = (2-P)÷(1-P)*.5
KURTOSIS = 12+(P*2)÷1-P
RANGE = {0,1,2,...}
```

Program: *PGEOMETRIC1* Group: *ANALYTIC*

Syntax: *R←PGEOMETRIC1 F* Subroutines: *NUNIFORM, PBETA PGAMMA, PNEGBIN PPOISSON, ΔPOWER*

Description: Returns samples from the geometric distribution.

Input: *F* = (1-*P*)÷*P*, where *P* is the probability of success on each trial

Output: *R* = The number of failures before the first success in a sequence of independent trials

Notes: *PGEOMETRIC1* is similar to *PGEOMETRIC* except the argument of *PGEOMETRIC1* is (1-*P*)÷*P* whereas the argument of *PGEOMETRIC* is *P*. The same conventions about argument and result length apply to *PGEOMETRIC1* as apply to *PGEOMETRIC*. See the documentation for *PGEOMETRIC* for complete details about the geometric distribution.

Program: *PHARMONIC* Group: *ANALYTIC*

Syntax: *R←PHARMONIC N* Subroutines: None

Description: Returns samples from Haight's discrete harmonic distribution with parameter(s) *N*.

Input: *N* = Distribution parameter. *N* can be a vector of (varying) positive integers.

Output: *R* = Samples from the distribution. If *N* is a vector, each element of *R* is drawn from the corresponding distribution.

Notes: For positive integer(s) *N*, the distribution is defined as the nearest integer to the ratio of *N* to an integer drawn at random from 1 to 2×*N*. The range of the distribution is a subset of {1,2,3,...,*N*} that includes 1 and *N*.

Program: *PHYPER* with *PGEO* — Group: *ANALYTIC*

Syntax: *R←N PHYPER A PGEO M* — Subroutines: None

Description: The combination of the programs *PHYPER* and *PGEO* returns samples from hypergeometric distribution(s).

Input: *N* = Sample size
M = Population size
A = Number of special items in *M*

The parameters *N*, *M*, and *A* can be vectors of equal length, or any subset of the parameters can be scalars.

Output: *R* = Samples from the distribution. An individual sample in *R* comes from the hypergeometric distribution with corresponding parameters in *N*, *A*, and *M*. The length of *R* depends on the length of *N*, *A*, and *M*.

Notes: The hypergeometric distribution is a discrete distribution which represents the number *X* of special items in a sample size of *N*, when there are a total of *A* special items in a population of size *M*. The hypergeometric distribution is therefore analogous to the binomial distribution, except sampling is done without replacement. The probability function is

```
F(X) = ((X!A)×(N-X)!M-A)÷N!M for X = 0,1,2,...,N⌊A
```

Selected statistics are

```
MEAN = N×A÷M
VARIANCE = N×(A÷M)×((M-A)÷M)×(M-N)÷M-1
SKEW =
         ((M-2×A)÷M)           ×  (((M-1)÷M-N)*.5)×(M-2×N)
  (N×(A÷M)×(M-A)÷M)*.5                    M-2
RANGE = {0,1,2,...,N⌊A}
```

Program: *PGEO* — Group: *ANALYTIC*

See the documentation for the program *PHYPER* for information about how to use *PGEO* in combination with *PHYPER* to generate samples from hypergeometric distributions.

Program: *PLAPLACE* Group: *ANALYTIC*

Syntax: *R←M PLAPLACE W* Subroutines: *NLAPLACE*, *NUNIFORM*

Description: Returns samples from the Laplace distribution(s).

Input: *M* = Scalar or vector representing the mean parameter
W = Scalar or vector representing the width parameter

If *M* and *W* are both vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to its respective distribution parameters. The length of *R* depends on the lengths of *M* and *W*.

Notes: The density function for the Laplace distribution with parameters *M*,*W* is

```
F(X) = (.5÷W)×*-|(X-M)÷W
```

Selected statistics are

```
MEAN = M
VARIANCE = 2×W*2
MEDIAN = M
MODE = M
SKEW = 0
KURTOSIS = 6
RANGE = (-INFINITY,INFINITY)
```

Program: *NLAPLACE* Group: *ANALYTIC*

Syntax: *R←NLAPLACE N* Subroutine: *NUNIFORM*

Description: Returns *N* samples from the "standard" Laplace distribution with density function

```
F(X) = .5×*-|X
```

Input: *N* = Number of samples to return

Output: *R* = Samples from the distribution

Notes: This distribution can be generalized by the transformation

```
Z←M+W×NLAPLACE N
```

and the random variable Z has

```
MEAN = M
VARIANCE = 2×W*2
MEDIAN = M
MODE = M
SKEW = 0
KURTOSIS = 6
RANGE = (-INFINITY,INFINITY)
```

See the documentation for *PLAPLACE*.

Program: *PLDECR* Group: *ANALYTIC*

Syntax: *R←PLDECR A* Subroutines: *NUNIFORM*, *PTRIANGULAR*

Description: Returns samples from the linear decreasing probability distribution(s).

Input: *A* = Scalar or vector parameter

Output: *R* = Samples from the distribution. An individual sample in *R* is chosen from the linear decreasing distribution with the corresponding parameter in *A*. The length of *R* is the length of *A*.

Notes: The density function for this distribution is a straight line of height 2÷*A* when *X*=0 that decreases to 0 when *X*=*A*, *A*>0. That is,

```
F(X) = 2×(A-X)÷A*2 for 0≤X≤A.
```

Selected statistics are

```
MEAN = A÷3
VARIANCE = (A*2)÷18
MODE = 0
MEDIAN = A×1-.5*.5
SKEW = .4×2*.5 = .5657
KURTOSIS = 2.4
RANGE = [0,A]
```

This basic distribution can be generalized by the transformation

```
Z←LOW+PLDECR HIGH-LOW
```

and the random variable *Z* has

```
MEAN = LOW+(HIGH-LOW)÷3
VARIANCE = ((HIGH-LOW)*2)÷18
MODE = LOW
MEDIAN = LOW+(HIGH-LOW)×1-.5*.5
SKEW = .4×2*.5 = .5657
KURTOSIS = 2.4
RANGE = [LOW,HIGH]
```

Program: *PLINCR* Group: *ANALYTIC*

Syntax: *R←PLINCR A* Subroutines: *NUNIFORM*, *PTRIANGULAR*

Description: Returns samples from the linear increasing probability distribution(s).

Input: *A* = Scalar or vector parameter

Output: *R* = Samples from the distribution. An individual sample in *R* is chosen from the linear increasing distribution with the corresponding parameter in *A*. The length of *R* is the length of *A*.

Notes: The density function for this distribution is a straight line of height 0 when *X*=0 that increases to 2÷*A* when *X*=*A*, *A*>0. That is,

```
F(X) = 2×X÷A*2 for 0≤X≤A.
```

Selected statistics are

```
MEAN = 2×A÷3
VARIANCE = (A*2)÷18
MODE = A
MEDIAN = A×.5*.5
SKEW = ¯.4×2*.5 = ¯.5657
KURTOSIS = 2.4
RANGE = [0,A]
```

This basic distribution can be generalized by the transformation

```
Z←LOW+PLINCR HIGH-LOW
```

and the random variable *Z* has

```
MEAN = LOW+2×(HIGH-LOW)÷3
VARIANCE = ((HIGH-LOW)*2)÷18
MODE = HIGH
MEDIAN = LOW+(HIGH-LOW)×.5*.5
SKEW = ¯.4×2*.5 = ¯.5657
KURTOSIS = 2.4
RANGE = [LOW,HIGH]
```

Program: *PLOGISTIC* Group: *ANALYTIC*

Syntax: *R←M PLOGISTIC W* Subroutines: *NLOGISTIC*, *NUNIFORM*

Description: Returns samples from the logistic distribution(s).

Input: *M* = Scalar or vector mean parameter
W = Scalar or vector width parameter

If both *M* and *W* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to its respective distribution parameters. The length of *R* depends on the length of *M* and *W*.

Notes: The density function for the logistic distribution with parameters *M*,*W* is

```
F(X) = (*-(X-M)÷W)÷W×(1+*-(X-M)÷W)*2
```

Selected statistics are

```
MEAN = M
VARIANCE = ((○W)*2)÷3
MEDIAN = M
MODE = M
SKEW = 0
KURTOSIS = 4.2
RANGE = (-INFINITY,INFINITY)
```

Program: *NLOGISTIC* Group: *ANALYTIC*

Syntax: *R←NLOGISTIC N* Subroutine: *NUNIFORM*

Description: Returns *N* samples from the "standard" logistic distribution with density function

F(*X*) = (*-*X*)÷(1+*-*X*)*2

Input: *N* = Number of samples to return

Output: *R* = Samples from the distribution

Notes: This distribution can be generalized by the transformation

Z←M+W×NLOGISTIC N

and the random variable Z has

MEAN = *M*
VARIANCE = ((○*W*)*2)÷3
MEDIAN = *M*
MODE = *M*
SKEW = 0
KURTOSIS = 4.2
RANGE = (*-INFINITY*,*INFINITY*)

See the documentation for *PLOGISTIC*.

Program: *PNORMAL* Group: *ANALYTIC*

Syntax: *R←M PNORMAL S* Subroutines: *NNORMAL, NUNIFORM*

Description: Returns samples from the normal distribution(s).

Input: *M* = Scalar or vector, the mean of the distribution

S = Scalar or vector, the standard deviation of the distribution

If both *M* and *S* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to its respective mean in *M* and standard deviation in *S*. The length of *R* depends on the length of *M* and *S*.

Notes: The density function for the normal distribution with mean *M* and standard deviation *S* is

```
F(X) = (÷S×(○2)*.5)×*¯.5×((X-M)÷S)*2
```

Selected statistics are

```
MEAN = M
VARIANCE = S*2
MEDIAN = M
MODE = M
SKEW = 0
KURTOSIS = 3
RANGE = (-INFINITY,INFINITY)
```

Program: *NNORMAL* Group: *ANALYTIC*

Syntax: *R←NNORMAL N* Subroutine: *NUNIFORM*

Description: Returns *N* samples from the standard normal distribution; that is, the distribution with density function

```
F(X) = (÷(○2)*.5)×*¯.5×X*2
```

Input: *N* = Number of samples to return

Output: *R* = Samples from the distribution

Notes: This basic distribution can be generalized by the transformation

```
Z←M+S×NNORMAL N
```

Selected statistics for *Z* are

```
MEAN = M
VARIANCE = S*2
MODE = M
MEDIAN = M
SKEW = 0
KURTOSIS = 3
RANGE = (-INFINITY,INFINITY)
```

See the documentation for *PNORMAL*.

Program: *PLOGNORMAL* Group: *ANALYTIC*

Syntax: *R←M PLOGNORMAL S* Subroutines: *NNORMAL*, *NUNIFORM*

Description: Returns samples from the lognormal distribution(s).

Input: *M* = Scalar or vector representing the mean
S = Scalar or vector representing the standard deviation

If both *M* and *S* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* comes from the lognormal distribution with the corresponding mean and standard deviation. The length of *R* is determined by the length of *M* and *S*.

Notes: The density function for the lognormal distribution is

```
F(X) = (÷X×D×(○2)*.5)××-(((⍟X)-A)*2)÷2×D*2
```

for `X>0`, where `A←⍟M÷(1+(S÷M)*2)*.5` and `D←(⍟1+(S÷M)*2)*.5` are the mean and standard deviation of an associated normal distribution.

Selected statistics are

```
MEAN = M
VARIANCE = S*2
MEDIAN = M÷(1+(S÷M)*2)*.5
MODE = M÷(1+(S÷M)*2)*1.5
RANGE = (0,INFINITY)
```

The definition of the lognormal distribution is that its logarithm is normal; that is, `⍟M PLOGNORMAL S` is a normal distribution with mean *A* and standard deviation *D*. If the mean and standard deviation of the associated normal distribution is known instead of the mean and standard deviation of the lognormal distribution, the program *PLOGNORMAL1* should be used. See the documentation for *PLOGNORMAL1*.

Program: *PLOGNORMAL1* Group: *ANALYTIC*

Syntax: *R←M PLOGNORMAL1 S* Subroutines: *NNORMAL*, *NUNIFORM*

Description: Returns samples from the lognormal distribution(s).

Input: *M* = Scalar or vector representing the mean of the associated normal distribution

S = Scalar or vector representing the standard deviation of the associated normal distribution

If both *M* and *S* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to its respective parameters in *M* and *S*. The length of *R* is determined by the length of *M* and *S*.

Notes: The density function for the lognormal distribution (one whose logarithm is normal) is

```
F(X) = (÷X×S×(○2)*.5)××-(((⍟X)-M)*2)÷2×S*2
```

for *X*>0. Selected statistics are

```
MEAN = *M+.5×S*2
VARIANCE =(*(2×M)+S*2)×¯1+*S*2
MEDIAN = *M
MODE = *M-S*2
RANGE = (0,INFINITY)
```

If the mean and standard deviation of the lognormal distribution are known instead of the mean and standard deviation of the associated normal distribution, the program *PLOGNORMAL* should be used. See the documentation for *PLOGNORMAL*.

Program: *PPASCAL* Group: *ANALYTIC*

Syntax: *R←N PPASCAL P* Subroutines: *NUNIFORM, PBETA PGAMMA, PNEGBIN, PPOISSON, ∆POWER*

Description: Returns samples from the Pascal (negative binomial) distribution(s).

Input: *N* = Scalar or vector; number of successes in a sequence of independent trials

P = Scalar or vector; probability of success on each trial

If both *N* and *P* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* comes from the Pascal distribution with corresponding parameters in *N* and *P*. The length of *R* depends on the length of *N* and *P*.

Notes: The Pascal distribution is a discrete distribution that can be interpreted as the number *X* of failures occurring prior to *N* successes in a sequence of independent trials where *P* is the probability of success on each trial. Thus *N + N PPASCAL P* is the total waiting time to achieve *N* successes. The probability function of the Pascal distribution is

```
F(X) = (X!N+X-1)×(P*N)×(1-P)*X
```

for *X* = 0,1,2,...

Selected statistics are

```
MEAN = N×(1-P)÷P
VARIANCE = N×(1-P)÷P*2
MODE = ⌊M←(N-1)×(1-P)÷P as well as
       M-1 if M is an integer
SKEW = (2-P)÷(N×1-P)*.5
KURTOSIS = 3+((P*2)+6×1-P)÷N×1-P
RANGE = {0,1,2,...}
```

See the documentation for *PNEGBIN* for a slightly different approach to the Pascal distribution.

Program: *PNEGBIN* Group: *ANALYTIC*

Syntax: *R←N PNEGBIN F* Subroutines: *NUNIFORM*, *PBETA* *PGAMMA*, *PPOISSON*, *ΔPOWER*

Description: Returns samples from the negative binomial (Pascal) distribution(s).

Input: *N* = Number of successes in a sequence of independent trials

F = (1-*P*)÷*P*, where *P* is the probability of success on each trial

Output: *R* = Number of failures occurring prior to *N* successes in a sequence of independent trials

Notes: *PNEGBIN* is similar to *PPASCAL* except the right argument of *PNEGBIN* is (1-*P*)÷*P* whereas the right argument of *PPASCAL* is *P*. The same conventions about argument and result length apply to *PNEGBIN* as apply to *PPASCAL*. See the documentation for *PPASCAL* for complete details about the negative binomial distribution.

Program: *PORDER* Group: *ANALYTIC*

Syntax: *R←N PORDER S* Subroutine: *NUNIFORM*

Description: Returns lowest or highest observations when sampling from the uniform distribution on the interval (0,1).

Input: *N* = Number of extremal observations desired
S = Number of observations in the sample, or vector of varying sample sizes

Output: *R* = Lowest *N* observations (if *N*>0) or highest |*N* observations (if *N*<0) in a sample of size *S*. If *S* is a vector of varying samples sizes, *R* is a |*N* by ρ*S* matrix whose columns contain |*N* observations for the respective sample sizes.

Program: *PPOISSON* Group: *ANALYTIC*

Syntax: *R←PPOISSON M* Subroutine: *NUNIFORM*

Description: Returns samples from the Poisson distribution(s).

Input: *M* = Scalar or vector representing the mean of the distribution

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to its respective mean in *M*. The length of *R* is the length of *M*.

Notes: The Poisson distribution with parameter *M* is a discrete distribution whose probability function is

```
F(X) = ((*-M)×M*X)÷!X for X = 0,1,2,... and M>0.
```

Selected statistics are

```
MEAN = M
VARIANCE = M
MODE = ⌊M as well as
       M-1 if M is an integer
SKEW = M*¯.5
KURTOSIS = 3+1÷M
RANGE = {0,1,2,...}
```

Program: *PSWITCH* Group: *ANALYTIC*

Syntax: *R←V PSWITCH P* Subroutine: *NUNIFORM*

Description: Returns samples from the "switch" distribution; a discrete, two-valued distribution that takes on value *V1* with probability *P* and takes on value *V2* with probability *1-P*.

Input: *V* = Two-element vector (*V1*,*V2*) or two-column matrix with each column containing several values for *V1* and *V2*

P = Probabilities. If *V* is a vector, *P* must also be a vector. If *V* is a matrix, *P* can be a scalar or a vector with one probability for each row of *V*.

Output: *R* = Samples from the switch distribution with parameters *V1*, *V2*, and the corresponding probability in *P*. An individual sample in *R* corresponds to the respective parameters in *V* and *P*. The length of *R* depends on the length of *V* and *P*.

Notes: The probability function for the switch distribution with parameters *V1*,*V2*,*P* is

```
F(X) = /   P  if  X = V1
       \ 1-P  if  X = V2
```

Selected statistics are

```
MEAN = (P×V1)+(1-P)×V2
VARIANCE = ((V1-V2)*2)×P×1-P
MODE = V1 if P ≥ .5
       V2 if P ≤ .5
SKEW = (P×(1-P)×1-2×P)÷(P×1-P)*1.5
KURTOSIS = (1+(¯3×P)+3×P*2)÷P×1-P
RANGE = {V1,V2}
```

Program: *PT* Group: *ANALYTIC*

Syntax: *R←PT DF* Subroutines: *NNORMAL*, *NUNIFORM*, *PBETA* *PCHISQUARED*, *PGAMMA*, *∆POWER*

Description: Returns samples from Student's t-distribution(s).

Input: *DF* = Scalar or vector; degrees of freedom

Output: *R* = Samples from the distribution. An individual sample in *R* comes from the t-distribution with the corresponding number of degrees of freedom in *DF*. The length of *R* is the length of *DF*.

Notes: The density function of the t-distribution with *DF* degrees of freedom is

```
F(X)=                G((DF+1)÷2)
      ──────────────────────────────────────────────
      ((○DF)*.5×G(DF÷2)×(1+(X*2)÷DF)*(DF+1)÷2
```

where *G*(•) is the gamma function.

Selected statistics are

MEAN = 0
VARIANCE = *DF*÷*DF*-2 if *DF*>2
MEDIAN = 0
MODE = 0
SKEW = 0
KURTOSIS = 3+6÷*DF*-4 if *DF*>4
RANGE = (-*INFINITY*,*INFINITY*)

Program: *PNONCENTRALT* Group: *ANALYTIC*

Syntax: *R←M PNONCENTRALT DF* Subroutines: *NNORMAL NUNIFORM*, *PBETA PCHISQUARED*, *PGAMMA*, *∆POWER*

Description: Returns samples from the non-central t-distribution(s).

Input: *M* = Scalar or vector representing the constant mean parameter

DF = Scalar or vector representing degrees of freedom

If both *M* and *DF* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to the respective elements in *M* and *DF*. The length of *R* depends on the lengths of *M* and *DF*.

Notes: This distribution is defined as the ratio of a normal distribution with mean *M* and variance 1, and a distribution (*X*÷*DF*)*.5 where *X* has the chi-square distribution with *DF* degrees of freedom, and is independent of the normal random variable in the numerator.

When *DF* is large, the non-central t-distribution (*M*,*DF*) is approximately normal with mean *M* and variance 1.

When *M* is 0, samples from the (central) t-distribution are produced.

Program: *PTRIANGULAR* Group: *ANALYTIC*

Syntax: *R←A PTRIANGULAR B* Subroutine: *NUNIFORM*

Description: Returns samples from the triangular distribution(s).

Input: *A* = Non-negative scalar or vector
B = Non-negative scalar or vector

If both *A* and *B* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* comes from the triangular distribution with the corresponding parameters in *A* and *B*. The length of *R* is determined by the lengths of *A* and *B*.

Notes: The density function for the triangular distribution with parameters *A,B* is a triangle that intercepts the *X*-axis when *X*=-*A* and *X*=*B*, and that rises to a peak of 2÷*A*+*B* when *X*=0. That is,

```
F(X) = / (2×X+A)÷A×A+B when -A≤X≤0
       \ (2×B-X)÷B×A+B when 0≤X≤B
```

Selected statistics are

```
MEAN = (B-A)÷3
VARIANCE = ((A*2)+(A×B)+B*2)÷18
MODE = 0
MEDIAN = (×B-A)×M-(.5×M×A+B)*.5 where M=A⌈B
SKEW =
  (.2×2*.5)×((2×B*3)+(3×A×B*2)+(¯3×B×A*2)+¯2×A*3)
  -----------------------------------------------
             ((A*2)+(A×B)+B*2)*1.5
KURTOSIS = 2.4
RANGE = [-A,B]
```

This basic distribution can be generalized by the transformation

```
Z←LIKELY+(LIKELY-LOW) PTRIANGULAR HIGH-LIKELY
```

and the random variable *Z* has

```
MEAN = (LOW+HIGH-2×LIKELY)÷3
VARIANCE = VARIANCE ABOVE, where A=LIKELY-LOW and
           B=HIGH-LIKELY
MODE = LIKELY
MEDIAN = LIKELY+MEDIAN ABOVE
SKEW = SKEW ABOVE
KURTOSIS = 2.4
RANGE = [LOW,HIGH]
```

Program: *PUNIFORM* Group: *ANALYTIC*

Syntax: *R←LOW PUNIFORM HIGH* Subroutine: *NUNIFORM*

Description: Returns samples from the uniform distribution(s).

Input: *LOW* = Scalar or vector representing the lower limit
HIGH = Scalar or vector representing the upper limit

If both *LOW* and *HIGH* are vectors, their lengths must conform.

Output: *R* = Samples from the distribution. An individual sample in *R* corresponds to its respective limits in *LOW* and *HIGH*. The length of *R* depends on the lengths of *LOW* and *HIGH*.

Notes: The density function for the uniform distribution with limits *LOW,HIGH* is

```
F(X) = 1÷HIGH-LOW for LOW≤X≤HIGH
```

Selected statistics are

```
MEAN = (LOW+HIGH)÷2
VARIANCE = ((HIGH-LOW)*2)÷12
MEDIAN = (LOW+HIGH)÷2
MODE (None)
SKEW = 0
KURTOSIS = 1.8
RANGE = [LOW,HIGH]
```

Program: *NUNIFORM* Group: *ANALYTIC*

Syntax: *R←NUNIFORM N* Subroutines: None

Description: Returns samples from the "standard" uniform distribution.

Input: *N* = Number of equally likely values from the interval [0,1].

Output: *R* = Samples from the distribution

Notes: The density function is

F(X) = 1 for 0≤*X*≤1

This basic distribution can be generalized by the transformation

Z←LOW+(HIGH-LOW)×NUNIFORM N

Selected statistics for *Z* are

MEAN = (*LOW*+*HIGH*)÷2
VARIANCE = ((*HIGH-LOW*)*2)÷12
MEDIAN = (*LOW*+*HIGH*)÷2
MODE (None)
SKEW = 0
KURTOSIS = 1.8
RANGE = [*LOW*,*HIGH*]

See the documentation for *PUNIFORM*.

Program: *PWEIBULL* Group: *ANALYTIC*

Syntax: *R←PWEIBULL P* Subroutine: *NUNIFORM*

Description: Returns samples from the Weibull distribution(s).

Input: *P* = Scalar or vector power parameter

Output: *R* = Samples from the distribution. An individual sample in *R* comes from the Weibull distribution with corresponding parameter in *P*. The length of *R* is the length of *P*.

Notes: The density function of the Weibull distribution with power *P* is

```
F(X) = P×(X*P-1)×*-X*P for X≥0
```

This basic distribution is the *P*th root of the exponential distribution with mean 1. It can be generalized by the transformation

```
Z←LOW+WIDTH×PWEIBULL P
```

and the random variable *Z* has

```
MEAN = LOW+WIDTH×G(1+1÷P)
        where G(•) is the gamma function
VARIANCE = (WIDTH*2)×G(1+2÷P)-(G(1+1÷P))*2
RANGE = (LOW,INFINITY)
```

Program: *TDIST* Group: *DISTFN*

Syntax: *R←DF TDIST X* Subroutines: None

Description: Computes Student's t-distribution.

Input: *X* = Vector of t-scores (greater than or equal to 0)

DF = Number of degrees of freedom (a single positive integer)

Output: *R* = Vector of probabilities matching the length of *X*. *R[I]* equals *PROB(-X[I]≤T≤X[I])* where *T* is the t-distribution with *DF* degrees of freedom.

Examples:

```
      10 TDIST 1.372 1.812 2.764
0.79994 0.89992 0.9800
```

Thus, for 10 degrees of freedom:

1.372 is the 90th percentile
1.812 is the 95th percentile
2.764 is the 99th percentile

```
      30 TDIST 1.31 1.697 2.457
0.79986 0.89995 0.97999
```

Similarly for 30 degrees of freedom.

Notes: Note that *PROB(0≤T≤X)* = .5 × *DF TDIST X*, and that *PROB(T≤X)* = .5 + .5 × *DF TDIST X* is the cumulative distribution function for *X*≥0.

CHAPTER 5

THE *STATISTICS* WORKSPACE

The *STATISTICS* workspace contains programs to calculate standard statistical measurements, derive correlation coefficients, and perform regression and residual analysis.

BASICSTATS 5-2
CORRELATE 5-5
REGRESSION 5-10

Program: *DSTAT* Group: *BASICSTATS*

Syntax: *DSTAT X* Subroutines: *FMT*, *STAT*

Description: Prints basic statistics for a data series.

Input: *X* = Data series

Output: Descriptive display of statistics

Example:

```
      DSTAT 15 12 11 16 17 14 8 19 13 8 11 10 9 5
SAMPLE SIZE     14
MAXIMUM         19
MINIMUM          5
RANGE           14
MEAN            12
VARIANCE        15.385
STD. DEVIATION   3.992
MEAN DEVIATION   3.143
MEDIAN          11.5
MODE             8      11
```

Program: *FREQ* Group: *BASICSTATS*

Syntax: *R←FREQ X* Subroutines: None

Description: Calculates frequencies for a discrete empirical distribution.

Input: *X* = Empirical data vector (discrete)

Output: *R* = Two-row matrix. The first row consists of the unique elements of *X* in ascending order, and the second row consists of the number of times the corresponding element in the first row occurs in *X*.

Example:

```
      FREQ 1 5 3 4 2 1 5 5 6 3
1 2 3 4 5 6
2 1 2 1 3 1
```

Program: *FREQDIST* Group: *BASICSTATS*

Syntax: *R←Y FREQDIST X* Subroutines: None

Description: Calculates the frequency distribution for "continuous" empirical data.

Input: *X* = Empirical data vector (from continuous population)

Y = Class specifications
- *Y*[1] = Left endpoint of left-most class
- *Y*[2] = Class width
- *Y*[3] = Number of classes

Output: *R* = Two-row matrix. The first row consists of the midpoints of each class in ascending order, and the second row consists of the number of elements of *X* lying in the corresponding class. (A class contains its right endpoint but not its left endpoint.)

Example: The program *NNORMAL* from the *PROBABILITY* workspace generates samples from the standard normal distribution.

```
      ¯3.5 1 7 FREQDIST NNORMAL 300
¯3 ¯2 ¯1   0  1  2  3
 7 17 78 106 74 15  3
```

(Actual output will vary from the illustration.)

Program: *PCTILES* Group: *BASICSTATS*

Syntax: *R←Y PCTILES X* Subroutines: None

Description: Calculates percentiles for a data vector.

Input: *X* = Data vector

Y = Percentiles desired. If *Y* is a single integer *N*, the *N*-1 percentiles are calculated. If *Y* is a vector of decimal values, the corresponding percentiles are calculated.

Output: *R* = Percentiles of the data *X* determined by *Y*. For every α,$(0 \le \alpha \le 1)$, the percentile corresponding to α is the smallest value in *X* which causes the empirical cumulative distribution function at *X* to be $\ge \alpha$.

Examples: The program *NNORMAL* from the *PROBABILITY* workspace returns the indicated number of samples from the standard normal distribution.

```
      X←NNORMAL 300

      4 PCTILES X
¯.6013 .0252 .6565

      .05 .1 .25 .5 .75 .9 .95 PCTILES X
¯1.5951 ¯1.1711 ¯.6013 .0252 .6565 1.2699 1.6362
```

(Actual output will vary from the illustration.)

Program: *ACORR* Group: *CORRELATE*

Syntax: *R←Y ACORR X* Subroutine: *ACOVAR*

Description: Calculates estimates of the autocorrelation function of a time series.

Input: *X* = Time series
Y = Largest autocorrelation lag desired (*Y*≤.5×length of *X*)

Output: *R* = Two-row matrix. The first row equals lags 1,2,3,...,*Y*; the second row equals the autocorrelation function at the corresponding lag.

Example:
```
      4 ACORR 1 3 2 5 3 7 5 10
 1         2          3         4
 0.1208    0.4333    ¯0.2292    0.05
```

Program: *ACOVAR* Group: *CORRELATE*

Syntax: *R←Y ACOVAR X* Subroutines: None

Description: Calculates an estimate of the autocovariance function of a time series.

Input: *X* = Time series
Y = Vector of lag indices for the autocovariance

Output: *R* = Autocovariance function of *X* at the given lags

Example:
```
      0 1 2 ACOVAR 1 3 2 5 3 7 5 10
 7.5 0.90625 3.25
```

Program: *CM* Group: *CORRELATE*

Syntax: *R←CM X* Subroutines: None

Description: Calculates a matrix of correlation coefficients.

Input: *X* = Matrix of observations; rows correspond to cases and columns correspond to variates.

Output: *R* = Matrix of correlation coefficients; *R[I;J]* is the correlation coefficient between variate *I* and variate *J*.

Example:

```
      X
 7 38 23 27
11  3 34 34
47 20 26 42
 2 40 27 34
 1 20  4  7
35 30 47 43
27  5 33 21

      CM X
 1         ¯0.2358    0.5322    0.6032
¯0.2358     1        ¯0.0799    0.2032
 0.5322    ¯0.0799    1         0.7683
 0.6032     0.2032    0.7683    1
```

Program: *CORR* Group: *CORRELATE*

Syntax: *R←V CORR M* Subroutines: None

Description: Calculates simple and partial correlation coefficients.

Input: *M* = Square matrix of simple correlation coefficients analogous to the result of the program *CM*.

V = Vector of variate numbers. The first two elements represent the pair of variates for which a coefficient is desired. Remaining elements designate variates whose effects should be removed in determining the partial correlation between variates *V*[1] and *V*[2].

Output: *R* = Simple (if 2=length of *V*) or partial correlation coefficient between variates *V*[1] and *V*[2].

Examples:

```
      X
 7 38 23 27
11  3 34 34
47 20 26 42
 2 40 27 34
 1 20  4  7
35 30 47 43
27  5 33 21

      ⎕←M←CM X
 1         ¯0.2358   0.5322   0.6032
¯0.2358    1        ¯0.0799   0.2032
 0.5322   ¯0.0799    1        0.7683
 0.6032    0.2032    0.7683   1
```

The following example is a simple correlation.

```
      2 3 CORR M
¯0.0799
```

The following example removes the effect of variate 1.

```
      2 3 1 CORR M
0.0554
```

The following example removes the effects of variates 1 and 4.

```
      2 3 1 4 CORR M
¯0.3577
```

Program: *PCORR* Group: *CORRELATE*

Syntax: *R←V PCORR X* Subroutine: *CM*

Description: Calculates simple and partial correlation coefficients.

Input: *X* = Matrix of observations. Rows correspond to cases and columns correspond to variates.

V = Vector of variate numbers. The first two elements represent the pair of variates for which a coefficient is desired. Remaining elements designate variates whose effects should be removed in determining the partial correlation between variates *V*[1] and *V*[2].

Output: *R* = Simple (if 2=length of *V*) or partial correlation coefficient between variates *V*[1] and *V*[2].

Examples:

```
      X
 7 38 23 27
11  3 34 34
47 20 26 42
 2 40 27 34
 1 20  4  7
35 30 47 43
27  5 33 21
```

This example is a simple correlation.

```
      2 3 PCORR X
¯0.0799
```

This example removes the effect of variate 1.

```
      2 3 1 PCORR X
0.0554
```

This example removes the effects of variates 1 and 4.

```
      2 3 1 4 PCORR X
¯0.3577
```

Notes: See also the program *CORR*.

Program: *CCORR* Group: *CORRELATE*

Syntax: *R*←*Y* *CCORR X* Subroutine: *CM*

Description: Calculates cross correlation.

Input: *X* = Two-column data matrix. The rows represent the observations of two variates.

Y = Positive integer specifying the maximum shift of the first variate relative to the second

Output: *R* = Two-row matrix. The first row contains the shifts from (-*Y*) to *Y* and the second row contains the correlation coefficients for the shifted first variate relative to the second. (Positive shifts occur when the first variate slides up relative to the second. The greater the shift, the fewer comparable observations can be used in the correlation.)

Example: *X*

```
68  14
 1  76
39  46
 7  54
42  22

      2 CCORR X
¯2          ¯1          0           1           2
¯0.3145     0.8584     ¯0.9205      0.8187     ¯0.8141
```

Note that ¯0.3145 is the coefficient when 68 1 39 is correlated with 46 54 22.

Program: *REG* Group: *REGRESSION*

Syntax: *T←V REG X* Subroutines: *DIV*, *DIVO*

Description: Performs simple and multiple regression.

Input: *X* = Matrix of observations. The rows correspond to cases; the columns correspond to variates.

V = Variate selection vector, consisting of some or all of the column indices of *X*. The last element of *V* indicates which variate is the dependent variable, and the remaining elements specify the independent variables.

Output: *R* = Five-column matrix containing various regression statistics. *R* consists of two submatrices:

|*A*|
|*B*|

The first row of *A* represents the intercept (constant term). The remaining rows of *A* correspond to selected variables. The columns of *A* correspond to the following:

1. Variable number (except *R*[1;1] is the dependent variable number)
2. Regression coefficient
3. Standard error of the regression coefficient
4. T-value for testing the hypothesis that the regression coefficient equals 0
5. Beta coefficient (except *R*[1;5] = 0)

The submatrix *B* has three rows and gives analysis of variance results. The rows of *B* refer to variation due to the regression, variation due to the error from the regression, and total variation. The columns of *B* correspond to the following:

1. 0
2. Degrees of freedom
3. Sum of squares
4. Mean square (except *B*[3;4] = Standard error of the estimate)
5. F-statistic (except *B*[2;5] = Multiple correlation coefficient *R* and *B*[3;5] = *R*⋆2)

Example: *DATA*

64	4	2
81	4	6
72	6	2
91	6	6
83	8	2
95	8	6

2 3 1 *REG DATA*

1	40.25	3.759	10.707	0
2	4.125	0.535	7.707	0.637
3	4	0.437	9.153	0.757
0	2	656.25	328.125	71.591
0	3	13.75	4.583	0.990
0	5	670	2.141	0.979

Notes: In the above example, the left argument of *REG* selects the following regression model:

*VARIATE*1 (*COL* 1 *OF DATA*) = *A* + (*B*×*VARIATE*2) + *C*×*VARIATE*3

The program calculates the coefficients *A* = 40.25, *B* = 4.125, and *C* = 4. The F-value (71.591) with (2,3) degrees of freedom shows that the regression is highly significant.

The result *T* of the program *REG* can also be used:

- as the right argument of the program *DSTATREG* to print a formatted display of the regression statistics, or
- as the left argument of the program *RES* to calculate the residuals for the regression.

Program: *DSTATREG* Group: *REGRESSION*

Syntax: *DSTATREG T* Subroutines: None

Description: Prints a report of the results of multiple regression.

Input: *T* = The matrix output of the *REG* program

Output: A descriptive display of the regression results. Variable 0 refers to the constant term (intercept).

Example: See the documentation for the program *REG* for the values of *T* and *DATA*.

```
      T←2 3 1 REG DATA
      DSTATREG T
```

VARIABLE	REG. COEFF.	S. E. COEFF.	T-VALUE	BETA COEFF.
0	40.250	3.759	10.707	
2	4.125	0.535	7.707	0.637
3	4.000	0.437	9.153	0.757

DEPENDENT VARIABLE = 1

	UNADJUSTED	ADJUSTED
MULTIPLE CORRELATION COEFF. R	0.990	0.987
R-SQUARED	0.979	0.974
STANDARD ERROR OF ESTIMATE	2.141	2.394

ANALYSIS OF VARIANCE

SOURCE OF VARIATION	DF	SUM SQUARES	MEAN SQUARE	F-VALUE
ATTRIBUTABLE TO REGRESSION	2	656.250	328.125	71.591
DEVIATION FROM REGRESSION	3	13.750	4.583	
TOTAL	5	670.000		

Program: *STEPREG* Group: *REGRESSION*

Syntax: *STEPREG X* Subroutine: *DIV*

Description: Generates user-controlled stepwise regression.

Input: *X* = Data matrix with the cases stored in the rows and the variates stored in the columns. The last column represents the dependent variable; all others are independent.

Output: The following global variables are produced after the final step:

VARIATES = ordered vector of variable (column) numbers used in the final step

COEFFICIENTS = vector consisting of the constant term (regression intercept) followed by one coefficient for each variate in the final step

RESIDUALS = vector of residuals from the regression at the final step

These residuals could be produced by the equation

```
X[;(ρX)[2]]-(1,X[;VARIATES])+.×COEFFICIENTS
```

The program automatically selects the variables to add or delete from the regression at each stage, based on partial F-value computations. A variable entered into the regression early can be removed later. Such variables are identified to you; you decide whether to accept the program's selections. If you choose not to delete a suggested variable, the program figures out which variable to add. If you choose not to add a suggested variable, or if the regression equation fits the data perfectly, the program ends. At each intermediate step, regression statistics and an analysis of variance are printed.

Example:

```
      DATA
2  3  4  7  9 10
1  3  7 11 15 17
3  4  6 10 10 16
2  3  8  8 11 11
1  3  5  9  9  9
```

When you execute *STEPREG DATA*, the program passes through the following variate selections if you agree with its selections: 4, 4 1, 4 1 5, 1 5, 5, 5 2, 5 2 3, 5 2 3 1. At that point a perfect regression relationship is found.

Program: *STREG* Group: *REGRESSION*

Syntax: *T←V STREG X* Subroutines: *CM*, *REG*, *DIV*, *DIVO*

Description: Performs simple stepwise regression.

Input: *X* = Matrix of observations. The rows correspond to cases; the columns correspond to variates.

V = Variate selection vector, consisting of some or all of the column indices of *X*. The last element of *V* specifies the dependent variable, and the remaining elements (in any order) specify the independent variables.

Output: *R* = Five-column matrix containing various regression statistics. *R* consists of two submatrices:

|*A*|
|*B*|

The first row of *A* represents the intercept (constant term). The remaining rows of *A* correspond to variables selected in decreasing order of the proportion of the variation of the dependent variable they represent. The columns of *A* correspond to the following:

1. Variable number (except *R*[1;1] is the dependent variable number)
2. Regression coefficient
3. Standard error of the regression coefficient
4. T-value for testing the hypothesis that the regression coefficient equals 0
5. Proportion of variation reduced (except *R*[1;5] = 0)

The submatrix *B* has three rows and gives analysis of variance results. The rows of *B* correspond to variation due to the regression, variation due to the error from the regression, and total variation. The columns of *B* correspond to

1. 0
2. Degrees of freedom
3. Sum of squares
4. Mean square (except *B*[3;4] = Standard error of the estimate)
5. F-statistic (except *B*[2;5] = Multiple correlation coefficient *R* and *B*[3;5] = *R*⋆2).

STREG requires more observations than variables selected in *V*. If a perfect regression is found, then the superfluous variates are not included in the result *T*.

Example:

```
          DATA
    150   47   50   36  200
    320   30   70  103  358
    100   26   50   50  167
     90   19   50   44  111
    260   38   70   61  290
     57    9   50   23   36
     68   27   50   30  133
     95   23   50   40  135
    248   67   90   55  399
    304   25   90 1024  279

          1 2 3 4 5 STREG DATA
    5   ¯60.961   51.967       1.173     0
    1     0.788    0.172       4.585     0.819
    2    ¯2.227    0.879       2.534     0.139
    4    ¯0.067    0.05        1.348     0.006
    3     1.274    1.401       0.909     0.005
    0    4  118636.598  29659.15       39.832
    0    5    3723.002    744.6         0.985
    0    9  122359.6       27.287       0.97
```

Notes: See also the documentation for the programs *STEPREG* and *REG*.

Program: *REGRESS* Group: *REGRESSION*

Syntax: *R←Y REGRESS X* Subroutines: *REG*, *DIVO*

Description: Performs simple regression for the model

$$Y = A + B \times X$$

Input: *X* = Independent variable
Y = Dependent variable

Output: *R* = Vector of regression statistics:
- *R*[1] = Intercept *A*
- *R*[2] = Coefficient *B*
- *R*[3] = Standard error of *B*
- *R*[4] = Computed t-value for *B*
- *R*[5] = Standard error of the estimate
- *R*[6] = Correlation coefficient
- *R*[7] = Square of the correlation coefficient

Example:

```
      10 12 15 20 28 40 57 REGRESS 10 20 25 30 35
 40 50
¯10.714 1.224 .194 6.311 6.283 .943 .888
```

Program: *RES* Group: *REGRESSION*

Syntax: *R←T RES X* Subroutines: None

Description: Calculates a table of residuals from a regression.

Input: *X* = Matrix of observations. Rows correspond to cases and columns correspond to variates.

T = Regression matrix output from the program *REG*.

Output: *R* = Four-column matrix. The first column contains the case number, the second column contains the observed value of the dependent variable, the third column contains the corresponding value estimated from the regression, and the fourth column contains the residuals.

Example:

```
      DATA
64  4  2
81  4  6
72  6  2
91  6  6
83  8  2
95  8  6

      T←2 3 1 REG DATA

      T RES DATA
1     64     64.75     ¯0.75
2     81     80.75      0.25
3     72     73        ¯1
4     91     89         2
5     83     81.25      1.75
6     95     97.25     ¯2.25
```

Notes: The output from *RES* can also be used as the right argument to the programs *STATRES* or *DSTATRES*. *STATRES* returns a vector of residual statistics, and *DSTATRES* returns a formatted display of these statistics.

Program: *STATRES* Group: *REGRESSION*

Syntax: *R←STATRES M* Subroutines: None

Input: *M* = Matrix output from the *RES* program or any matrix whose fourth column contains the residuals from some regression.

Output: *R* = Vector of statistics on the residuals:
- *R*[1] = Sum of the residuals
- *R*[2] = Sum of the absolute values of the residuals
- *R*[3] = Sum of the squares of the residuals
- *R*[4] = Durbin-Watson statistic
- *R*[5] = Number of runs of positive or negative residuals
- *R*[6] = Number of positive signs
- *R*[7] = Number of negative signs
- *R*[8] = Expected value of the number of runs
- *R*[9] = Standard deviation of the number of runs

Example: See the documentation for the program *REG* for the values of *T* and *DATA*; see the documentation for the program *RES* for the value of *M*.

```
      T←2 3 1 REG DATA
      M←T RES DATA
      M[;4]
¯0.75 0.25 ¯1 2 1.75 ¯2.25

      STATRES M
0 8 13.75 2.009090909 5 3 3 4 1.095445115
```

Program: *DSTATRES* Group: *REGRESSION*

Syntax: *DSTATRES M* Subroutine: *STATRES*

Description: Prints a report on the residuals from a regression.

Input: *M* = The matrix output of the *RES* program or any matrix whose fourth column contains the residuals from some regression.

Output: A descriptive display of statistics on the residuals.

Example: See the documentation for the program *REG* for the values of *T* and *DATA*; see the documentation for the program *RES* for the value of *M*.

```
      T←2 3 1 REG DATA
      M←T RES DATA
      M[;4]
¯0.75 0.25 ¯1 2 1.75 ¯2.25

      DSTATRES M
SUM OF RESIDUALS                  0
SUM OF ABSOLUTE RESIDUALS         8
SUM OF SQUARES OF RESIDUALS       13.75
DURBIN-WATSON STATISTIC           2.009090909
NUMBER OF RUNS                    5
   POSITIVE SIGNS                 3
   NEGATIVE SIGNS                 3
EXPECTED NUMBER OF RUNS           4
   STANDARD DEVIATION             1.095445115
```

INDEX OF PROGRAMS AND GROUPS

ACCSF 2-50
ACOMPARE 2-6
ACORR 5-5
ACOVAR 5-5
ACRS 2-44
AFDIFF 2-7
AGGREGATE 2-15
AGR 3-9
ANALYTIC (Group) 4-7
ANNUITY 2-97
AROR 2-78
ASSETS (Group) 2-2
ASYMPTOTE 3-13
AVEBAL 2-8
AVGCOST 2-58
AVGRED 2-8

ΔBAL 2-21
BALFOR 2-18
ΔBALS 2-21
BASICSTATS (Group) 5-2
BCMATX 2-66
BOND 2-4
BONDS (Group) 2-4
BPRICE 2-5
BREAKEVEN 2-19
BRMATX 2-66
BYIELD 2-5

CALCULATE (Group) 2-6
CASH (Group) 2-18
CASHMGT 2-2
CCORR 5-9
CDCF 2-29
CFV 2-32
CHANGETIME (Group) 2-22
CIDF 2-31
CIROR 2-30
CM 5-6
CONTRATES (Group) 2-29
CORI 2-61
CORR 5-7
CORRELATE (Group) 5-5
CPV 2-33
CTIMEVALUE (Group) 2-32
CUM 2-9
CUMDISCRETE 4-2
CUMHISTOGRAM 4-6
CUMPROD 2-9
CUMRANGE 2-93
CUMZEROMAX 2-94
CURRENT 2-20
CYCLE (Group) 3-2

DAYLAG 2-52
DBAL 2-48
DCF 2-72
DCIDF 2-86
DCIROR 2-83
DEBT (Group) 2-34
DEPRE 2-45
DEPRECIATION (Group) 2-44
DERIVETREND (Group) 3-4
DGROWTH 3-8
DIROR 2-79
DISCOUNT 2-60
DISCRETE 4-3
DISTFN (Group) 4-42
DISTRIBUTE (Group) 2-51
DIV 2-10
DIVRED 2-10
DLEVPAY 2-37
DORG 3-24
DPOWER 3-7
DSEASFACTOR 3-2
DSTAT 5-2
DSTATREG 5-12
DSTATRES 5-19
DSTEPTREND 3-6
DTTREND1 3-4
DTTREND2 3-5

ELTOF 2-61
EMPIRICAL (Group) 4-2
EPRIN 2-34
EXPOGROWTH 3-13
EXPOSMOOTH (Group) 3-10
EXPOSMOOTH1 3-10
EXPOSMOOTH2 3-11
EXPOSMOOTH3 3-12
EXPOTREND (Group) 3-13
EYIR 2-55

FDIFF 3-18
FEDTAX79 2-91
FGROWTH 3-17
FIFO 2-59
FIRST 2-25
FORETREND (Group) 3-15
FPOWER 3-16

FRATIO 3-19
FREGRESS 3-18
FREQ 5-2
FREQDIST 5-3
FTRENDRATIO 3-19
FTTREND1 3-15
FTTREND2 3-15
FV 2-73

GLOAN 2-35
GREATEST 2-11
GROWTH 3-24

HALFYR 2-26
HISTOGRAM 4-4

IDF 2-84
INCOMEASSETS 2-88
INCREASE 2-62
INIT 2-62
INTEREST (Group) 2-55
INTRANGE 2-94
INVENTORY (Group) 2-58
IROR 2-76
ITC 2-92

LAST 2-25
LEADLAG 2-51
LEAST 2-11
LEVPAY 2-36
LEVPAY1 2-39
LFF 2-64
LIFO 2-59
LOAN 2-40
LOFC 2-41
LZF 2-63

MANIPULATE (Group) 2-61
MATRIX (Group) 2-66
MAX 2-67
MAXHALT 2-96
MAXSCAN 2-68
MIDF 2-85
MIN 2-67
MINHALT 2-96
MINMAX (Group) 2-67
MINSCAN 2-68
MINUS 2-14
MIROR 2-77
MOVINGAVE 3-20
MOVMAXSCAN 3-20
MOVMINSCAN 3-21
MPROD 2-12
MQY 2-22
MQYFACTOR 2-24
MSAROR 2-80
MTW 2-26

NCAUCHY 4-11
NEXPONENTIAL 4-15
NEXTREME 4-7
NLAPLACE 4-23
NLOGISTIC 4-27
NNORMAL 4-29
NUNIFORM 4-40
NYIR 2-55

PAYMT 2-20
PBETA 4-8
PBINOMIAL 4-9
PCAUCHY 4-10
PCENTRALF 4-17
PCHISQUARED 4-12
PCOMPARE 2-6
PCORR 5-8
PCTILES 5-4
PCTOF 2-13
PDCF 2-69
PER 2-88
PEXPONENTIAL 4-14
PF 4-16
PFDIFF 2-7
PFV 2-70
PGAMMA 4-18
PGEO 4-21
PGEOMETRIC 4-19
PGEOMETRIC1 4-20
PHARMONIC 4-20
PHYPER 4-21
PLANT 2-3
PLAPLACE 4-22
PLDECR 4-24
PLINCR 4-25
PLOGISTIC 4-26
PLOGNORMAL 4-30
PLOGNORMAL1 4-31
PLUS 2-14
PLUSMINUS 2-65
PNEGBIN 4-33
PNON 4-17
PNONCENCHISQ 4-13
PNONCENTRALT 4-37
PNORMAL 4-28
POLYGON 4-5
POLYGONCUM 4-6
PORDER 4-33
POWER 3-23
PPASCAL 4-32
PPOISSON 4-34
PPV 2-71
PRIOR 2-27
PSWITCH 4-35
PT 4-36
PTIMEVALUE (Group) 2-69
PTRIANGULAR 4-38

PUNIFORM 4-39
PV 2-74
PWEIBULL 4-41

QFACTOR 2-53
QTR2NDINTEREST 2-56

RANGE 2-95
RATES (Group) 2-72
RATIOS (Group) 2-88
REG 5-10
REGRESS 5-16
REGRESSION (Group) 5-10
RELATION (Group) 3-18
REPLACE 2-62
RES 5-17
RLF 2-63
RND 2-89
ROR 2-75
ROUND 2-89
ROUNDDOWN 2-90
ROUNDING (Group) 2-89
ROUNDUP 2-90
RZF 2-63

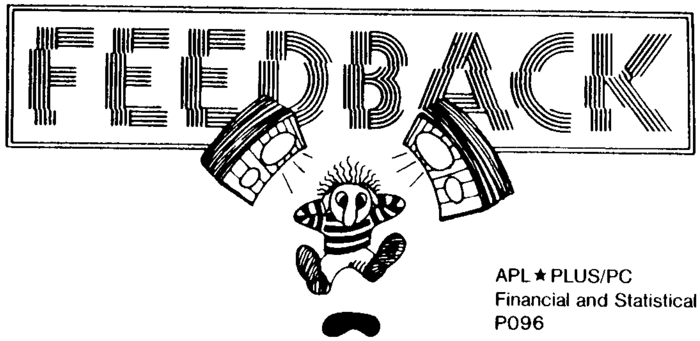
SAROR 2-81
SBALFOR 2-18
SCURVE 3-14
SEASFACTOR 3-3
SFCFL 2-87
SFROR 2-82
SFUND 2-49
SHIFT 2-28
SMOOTHING (Group) 3-20
SPREAD 2-54
STATRES 5-18
STEPREG 5-13
STEPTREND 3-23
STL 2-46
STREG 5-14
SYD 2-47

TAXES (Group) 2-91
TDIST 4-42
TESTS (Group) 2-93
TIMES 2-12
TIMEVALUE (Group) 2-97
TLOAN 2-42
TOTAL 2-15
TRANGE 2-95
TRANSPOSE 2-64
TREND (Group) 3-22
TTREND1 3-22
TTREND2 3-22

VLOAN 2-43
VOLCOST 2-60

WEIGHTEDAVE 3-21
WTDSUM 2-16
WTM 2-27

YFACTOR 2-53
YR2NDINTEREST 2-57
YTD 2-17



APL★PLUS/PC
Financial and Statistical Library
P096

We write our publications to help you. You can help us by filling out and returning FEEDBACK. Your reply will be forwarded to the person who can best benefit from your comments.

1. Is the writing straightforward and clear? Yes ☐ No ☐

2. Is the material organized well? Yes ☐ No ☐

3. Are the examples clear and well chosen? Yes ☐ No ☐

4. Do the illustrations (figures and tables) clarify the material? Yes ☐ No ☐

5. Do you find the size of the publication and the way it is bound convenient? Yes ☐ No ☐

6. Are there errors, or misleading information, in the publication? If so, where do they occur? Yes ☐ No ☐

7. Did you find any parts of the publication unduly hard to understand? If so, which are they? Yes ☐ No ☐

How can we change this publication to make it more useful to you? (If you need more room, please write on the back.)

Thank you for your comments. No postage is necessary if mailed in the U.S.A.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT #10596 WASHINGTON, D.C

Postage Will Be Paid By Addressee

STSC, Inc.
2115 East Jefferson Street
Rockville, Maryland 20852

ATTENTION: Marketing Communications