NASA Contractor Report 177986

FINDS: A Fault Inferring Nonlinear Detection System --

Programmer's Manual

Version 3.0

Roy E. Lancraft

BBN Laboratories Inc.
10 Moulton Street
Cambridge, Massachusets 02238

Contract NAS1-16579

December 1985

NASA

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

TABLE OF CONTENTS

CONTENTS

APPENDIX A   SUMMARY OF SPECIFIC HARDWARE AND SOFTWARE REQUIREMENTS

APPENDIX B   GENERATING THE FINDS PROGRAMMERS MANUAL

ii

LIST of FIGURES

LIST of TABLES

LIST of ABBREVIATIONS

A/C   Aircraft

ATOPS   Advanced Transport Operating Systems

Azm   MLS azimuth

B-frame body frame

BFF   Bias-Free Filter

DME   Distance Measuring Equipment

E-frame earth fixed rotating frame (Earth-frame)

EKF   Extended Kalman Filter

El   MLS elevation

FDI   Failure Detection and Isolation

FDIR   Failure Detection Isolation and Reconfiguration

FINDS   Fault Inferring Nonlinear Detection System (computer program)

FTN   Fault Tolerant Navigator

FTS   Fault Tolerant System

G&C   Guidance and Control

G-frame geographic frame located at the runway

I-frame earth centered nonrotating frame (Inertial-frame)

IAS   Indicated Airspeed

IC's   Initial Conditions

IMU   Inertial Measurement Unit

L-frame vehicle carried (N.E.D) frame (Local Level frame)

LRT   Likelihood Ratio Test

ML   Maximum Likelihood

MLS   Microwave Landing System

MTBF   Mean Time Between Failures

NFF   No-Fail Filter

P,Q,R   body rate gyros

RA   Radar Altimeter

Rng   MLS range

RSDIMU Dual Fail-Operational Redundant Strapdown Inertial Measurement
        Unit

TSRV   Transport Systems Research Vehicle

1 INTRODUCTION

This report provides detailed software documentation of the digital computer program FINDS (Fault Inferring Nonlinear Detection System) version 3.0. FINDS is a highly modular and extensible computer program designed to monitor and detect sensor failures, while at the same time providing reliable state estimates. In this version of the program the FINDS methodology is used to detect, isolate and compensate for failures in simulated avionics sensors used by the Advanced Transport Operating Systems (ATOPS) Transport Systems Research Vehicle (TSRV) in a Microwave Landing System (MLS) enviornment. It is intended that this report serve as a programmers guide to aid in the maintenance, modification, and revision of the software.

Throughout this manual we have assumed that the reader has read and is familiar with the contents of the following reports:

1. FINDS: A Fault Inferring Nonlinear Detection System - User's Guide, NASA CR-172199, September 1983.

2. A Fault Tolerant System for an Integrated Avionics Sensor Configuration, NASA CR-3834, 1984.

3. An Aircraft Sensor Fault Tolerant System, NASA CR-165876, April 1982.

The primary goal of this manual is to provide in depth documentation of the current version of the FINDS software. To accomplish this goal, detailed descriptions are provided for the program's modules (functions, and subroutines) and their internal data structures (common blocks) developed by BBN. In addition, the contents and purpose of each disk file will be examined along with the steps required to rebuild the library and executable files used by FINDS. Detailed information about the program's external data structures (input and output files), as well as information about the program's overall structure and intended usage (from a users point of view) can be found in [1], and therefore will not be covered in this report. It should be clearly noted that NOT ALL functions and internal data structures used by FINDS will be described in this report -- instead only those which pertain to the simulation independent portion of the program will be considered. This approach was taken because the simulation environment in which FINDS operates was originally

developed and supplied by NASA-LRC, and therefore it was felt the emphasis of this document should only be on the newly developed software.

A secondary goal of this work is to provide a convienent mechanism for documentation information contained herein to be maintained and improved upon. Some of the problems associated with writing a programmers or users guide for a developmental computer program, such as FINDS, is that it a) is never quite comprehensive enough, and b) is obsolete soon after it is printed. This is true in part because developmental programs are never quite stable (i.e. they are constantly being modified as new provisions are added, or as "bugs" are found), and in part because incremental (i.e., as modules are written) documentation is seen by many to be both time consuming and fragmented - therefore it is not always done. This clearly confounds the development process itself, since only a few people know the "inner workings" of the program. In an effort to help alleviate some of these inherent problems, we have written this programmer's manual in such a way that it can be re-created semi-automatically from specially commented source code and text files. The goal was to make it easy to incorporate changes which occurred since the last time a manual was created. To accomplish this, special command files and programs were created to generate files which could be processed by the Digital Standard Runoff text formatting program. In addition, all the figures and tables used in the manual were generated on an Apple Lisa personal computer (using LisaDraw software) - so they too can be easily modified and re-generated to account for changes to the code.

The organization of this report is as follows: Chapter 2 consists of a comprehensive overview of the FINDS software, along with installation instructions. Chapter 3 provides detailed descriptions of the FINDS program modules, as well as an overview of some notational conventions used in the report. The internal data structures and a summary of the indexing schemes employed can be found in Chapter 4. Appendix A gives a list of specific hardware and software requirements (including a list of all supplied software). Appendix B contains the "rules" for formatting source files and a description of how this manual can be automatically re-generated. As a further aid, a cross-reference list of all file names, common block names, module names, and other key words documented in this report can be found at the end of the report.

The following suggested reading of the manual is encouraged:

General information and installation:

1,2,Appendix A

Complete reading:

1,2,3.1,4.1, remainder of Chapter 3, and 4, Appendix A, and Appendix B.

2 SOFTWARE OVERVIEW AND INSTALLATION DETAILS

This chapter describes how the software is organized from the vantage point of the VAX 11/780 operating system. A user's perspective on the functional organization and other aspects of the FINDS software and its' utility programs are provided in [1]. The chapter is organized in the following fashion: Section 2.1 gives an overview of the delivered software by reviewing the contents and intended purpose of each file supplied. (Note: For quick reference, Appendix A also provides a brief summary of these files.) Section 2.2 describes the steps nessesary to install (or rebuild) each of the programs. Automatic re-generation of the programmers guide is covered separately in Appendix B.

2.1 Software Overview

This section describes the contents and intended scope of each of the disk files which comprise FINDS and its associated utility programs. A complete list of all the delivered software, as well as the specific hardware requirements, are described in Appendix A. Detailed descriptions of the individual modules contained in each file can be found in the next chapter.

It is convenient to assume that the operational software is stored in a main directory which will be called the FINDS directory. The organization of this directory is straightforward. There are four executable programs in the directory - each requiring FOR, OLB, COM, and/or OPT files for their creation. The four executable files are detailed below:

DOC.EXE   –   A program to extract specially formatted and embedded documentation from Fortran (or Ratfor) source files (see Appendix B for a description of its use).

FINDS.EXE   –   FINDS version 3.0 simulation program documented in this manual and in [1].

PLOTD.EXE   –   Program to plot the time history output, generated by FINDS, on a Tektronix 4010/4014 or compatible terminal (see [1]).

PRINTD.EXE –   Program to print the time history output, generated by FINDS, in tabular form on either the users terminal or a disk file (see [1]).

The FINDS directory contains a single library file:

FINDSLIB.OLB-   Utility library built using the FINDS sources (FORTRAN
                files).

Several command and linker option files can be found in the FINDS
directory.   Command files (extension = COM) are used to automate the
building and maintainence of FINDS. As will be seen in the next
section, a by-product of using command files is that it simplifies
transporting the software to other VAX or users sub-directories.
Linker options files are used at link time to specify how to build an
executable image. The command and linker options files are summarized
below:

FINDSC.COM   - Properly compiles all the FORTRAN source files which
                  are used by FINDS.
FINDSL.COM   - Properly links together the object and library files
                  to produce FINDS.EXE.
FINDSLIB.COM -   Compiles the source files and builds the library file
                  FINDSLIB.OLB
GETDOC.COM   - Automatically gets the latest documentation
                  from all FINDS routines (see Appendix B for
                  more information).
FINDSPG.COM   - Automatically builds a new FINDS programmers
                  guide (see Appendix B).
PLOTD.OPT   - Linker options file for PLOTD
PRINTD.OPT   - Linker options file for PRINTD
FOREIGN.COM   - Establishes useful logical and symbolic names

2.2 Installation Notes

The following steps are required to initially install the FINDS
software:

1. Copy all files from magnetic tape onto a suitable VAX disk
    directory using the VAX/VMS Backup utility. Let's assume
    this directory is named "finds" for the subsequent
    discussions.

5

2. Edit the file "foreign.com" and correct the directory names referenced so that they point to directory [finds].

3. Type
        $ @foreign.com
    to install the logical names and symbols. These will be will be used subsequently. (This step can be made part of the user's login.com file if these symbols are used frequently.)

4. Compile all FORTRAN sources:
        $ @findsc
        $ compile plotd
        $ compile printd

5. Create the FINDS library file findslib.olb:
        $ @findslib

6. Create the executable files:
        $ @finds
        $ link plotd/opt
        $ link printd/opt

7. Generate all the required input data files required for running FINDS using the text editor. (See [1] for detailed directions on how to create these files.)

8. Run FINDS by typing:
        $ finds

9. Run the graphical analysis tool PLOTD by typing:
        $ plotd

10. Run the tabular examination tool PRINTD by typing:
        $ printd

Once the software has been installed, incremental changes can be made as follows:

1. Modify a Fortran source file. Be sure to update the embedded documentation.

2. Compile it. (e.g. $ compile filename)

3. Update the library file (this step is required for files futsub.for, fvmsub.for, and fiosub.for.)
           $ update filename

4. Re-build FINDS
           $ @finds

For instructions on how to generate and maintain the programmers guide see Appendix B.

3 MODULE DESCRIPTIONS

The following subsections contain detailed descriptions of FINDS routines, organized according to source files (refer to Chapter 2 for a list of supplied files). The first subsection reviews some of the notational shorthand used in the descriptions. The second subsection contains a brief description of the contents of each file, containing a statement of the name of the source file, a description of the nature of its contents, and then a list and short synopsis of each subroutine it contains. The remaining subsections contain more detailed descriptions of each subroutine - many of which have companion flowcharts. Each such description contains a statement of the subroutine function, a sample call, and a description of the required arguments in the form:

> name   type   in,out, or inout   units   description

These are followed by a list of all other routines called, all routines which reference it, and all common blocks used by the routine. Full descriptions of most of the common blocks can be found in Chapter 4.

3.1 Some Notational Conventions

In order to condense the textual descriptions and flowcharts we've adopted various shorthand notations. This section itemizes these conventions.

In specifying the argument descriptions we've assumed the following:

```
Variable type can be:
        . integer   -   integer*4
        . real      -   real*4
        . double    -   real*8
        . logical   -   logical*4
        . char      -   character*(x)
        . char*n    -   character*n
        . byte      -   logical*1

Units can be:
        . a standard engineering unit or
        . unitless -   no units (i.e. a cardinal or pointer
```

8

                              index)
    . mixed   —   various units (usually used for
                    vectors, matrices, and scratch areas)
    . temp   —   temporary, i.e. units vary
    . string   —   ASCII characters

Arrays (matrices and vectors) are usually specified by upper case names. Both upper and lower case are often used to aid in interpreting the mnemonic used. The following shorthand notation is used when discussing arrays or equations involving arrays:

    A(i,j)   —   the i,j'th element of the matrix A
    V(i)   —   the i'th element of the linear array V
    A[i]c   —   the i'th column of the array A
    A[i]r   —   the i'th row of the array A
    A[i:j]c   —   the submatrix comprised of the i'th through
                   j'th columns of the array A
    A[i:j]r   —   the submatrix comprised of the i'th through
                   j'th rows of the array A
    A*B   —   matrix multiplication of A and B
    a*b   —   scalar multiplication of a and b (Note: lower
                   case usually implies a scalar variable)

The flowcharts contained in this manual are not meant to be complete descriptions of the routines. Instead, they are intended to enhance the reader's understanding of the software's structure and to highlight the software techniques employed. As such, they should be used in conjunction with the written documentation and commented source code itself. For example, one particular flowchart may show, by detailed enumeration, how the internal data structures are used, whereas in an other case a top-level functional flowchart will be presented to highlight an important theme.

Most of the symbolic notation used in the flowcharts are described in Figure 1. Notation inside subroutine boxes may contain the following:

    . the box can contain the subroutine name,
    . the subroutine name and its arguments, or
    . the subroutine name and a key argument.

As a general rule, always refer to the written documentation for the correct calling sequence to use. Array indexing conventions are described in Chapter 4.

START
(ENTER)

The beginning of a process (subroutine)

STOP
(EXIT)

The End of a process (subroutine)
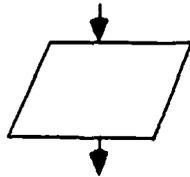
An I/O box
performs the indicated
input and/or output operations
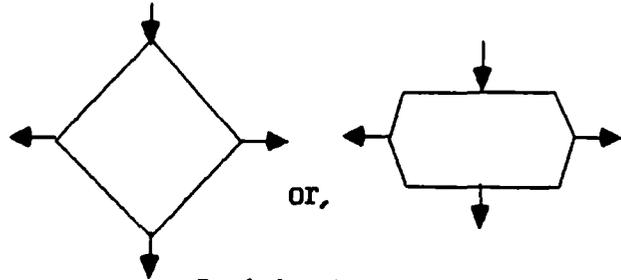
Decision boxes
exits are labeled and one route is
taken depending on the result of
the computation indicated in the
box

or,

An onpage connector
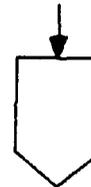
An offpage connector
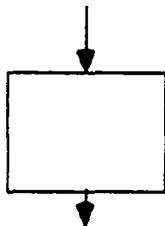
An instruction box
performs operations
called for in the box
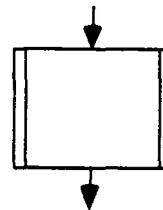
Subroutine box
performs operations via the
subroutine named in the box

Figure 1. Definition of Flow Diagram Symbols

3.2   Brief Summary Of Contents By Source File

name: FMAIN.FOR
cont: This file contains the main simulation program which
        orchestrates the execution of FINDS.
subr: FINDS: (program unit) Top level simulation program
        INITAL: initializes several basic simulation quantities
        SET: initializes constants (such as conversion
                 factors) used in FINDS

name: FSFDI.FOR
cont: This file contains all the "core" routines necessary to
        implement the FINDS fault tolerant navigator and FDIR
        software. As such, these routines roughly represent
        the simulation independent portion of the program.
subr: NAV -   fault tolerant navigator - orchestrates the operation
                 of FINDS FTN and FDIR functions
        INITG - general initialization for FINDS
        INITF - perform initialization specific to the no-fail filter
        STARTF- start-up procedure for the no-fail filter, i.e.
                 choose initial conditions
        SUMIN - forms the input vector to the no-fail filter
        SUMOUT- forms the measurement vector for the no-fail filter
        GYROCR- compute compensation for the rate gyros due to
                 the earth's rotation
        GTOI -   compute inertial quantities from ground based
                 estimates
        CKUNST- check the no-fail filter estimates for divergence
        KALMN - executive routine to implement an extended Kalman
                 filter using a bias filter decomposition
        EKFN1 - bias-free portion of the extended Kalman filter
        BIASF - bias portion of the extended Kalman filter
        BLEND - blend the bias and bias-free states and covariances
                 together to form the total no-fail filter estimates
        BLGAIN- compute the blender gain
        SETISN- update a count of the current number of sensors
                 used by the no-fail filter
        CHKRAD- check for switch over to radar altimeter, and

reconfigure the no-fail filter at switch over time

UPDA – update the discrete state transition matrix

UPDAB – update the discrete state transition matrix to include the coupling due to the biases

UPDB – update the discrete input matrix

UPDQ – update the discrete process noise covariance matrix

UPDH – update the non-linear measurement function

UPDPH – update the partial of $h[x(k)]$ w.r.t. $x(k)$

DETECT– implements a bank of detectors and likelihood ratio computers

LKF – first order linear Kalman filter – used to estimate a hypothesized failure's level

LRT – computes a log-likelihood ratio

DECIDE– performs failure decision functions

RECONF– reconfigures the FTS after failures and/or healings

CLPSIO– collapse (expand) the no-fail filter to reflect failure (healing) of a sensor

NOISR – reset elements in the no-fail filter process and measurement covariance matrices to reflect the loss (addition) of a sensor

RESCMP– compute the expanded residuals sequence from the (collapsed) innovations sequence produced by the no-fail filter

FILCOL– estimate colored MLS noise states (used to compensate the innovations sequence to account for its colored statistics)

CLPSBE– collapse (expand) the bias estimator to reflect the removal (addition) of a bias

ADSTBP– adjust (manage) pointer vectors used in the bias estimator

RCOV – reset the no-fail filter state estimation error covariance after FDI of a failure

MINSET– check to be sure filter will remain stable after a candidate sensor is removed

HEALR – monitor failed sensors and test for healing

AVECMP– compute the sum of the difference between two like sensors over the healing window

LRTHLR– compute a LRT for the healing of a sensor at the end of the healer windows

CONVRF– returns the conversion factor required to convert from program to user (printout) units for a particular no-fail filter state or sensor

AVBIAS– computes the average measurement bias as seen by the no-fail filter

name: FGAC.FOR

cont: This file contains routines used to simulate the aircraft
and the guidance and control logic used in the aircraft.
These routines were originally part of program FILCOMP.

subr: ACEQIN- integrate the aircraft equations of motion ahead
               one simultion step
       AUTLD - auto-land control laws
       AUTTHR- compute throttle commands
       BANKTR- RNAV guidance and control outputs i.e. commands
               to guide the aircraft before AUTLD takes over
       CNTRLS- generate the control signals using either true
               (i.e. simulated) or estimated quantities
       ESTPNP- compute estimates of waypoint quantities and store
               them in EWP
       RUDDER- dynamics for the rudder servo and yaw rate damper
       SERVO - elevator and aileron servo dynamics
       STABCN- stabilizer trim control logic
       THRUSD- engine thrust dynamics (accurate above 10 degrees
               throttle setting)
       WAYPNT- compute all data for waypoint segment planning

name: FWIND.FOR

cont: Contains routines used to simulate the wind and gust
environments to which the aircraft will be subjected.

subr: BREEZE- computes shear winds, calls WINDGT to generate gusts
               and sums the wind components to form the total winds
       GROUNE- computes the effects of ground proximity - called
               ground effects - as incremental terms added to pitch,
               lift, and drag
       WINDGT- generates gust components which are added to u,v,w
               and P,Q,R terms in the aircraft simulation

name: FSENS.FOR
cont: This file contains all the routines used to simulate the
        normal operation and the "failed" behavior of sensors and
        sensor sub-systems in FINDS. All sensors contained in
        this file can be simulated with up to triple redundancy -
        except for the RSDIMU. The reader can find detailed
        descriptions on how each sensor is simulated and how to
        modify the parameters of these modules in section 3.3 of [1].
subr: RADALS- radar altimeter sensor module
        AIRSPS- indicated airspeed sensor module
        BMRGS - flight quality body mounted rate gyro sensors (P,O,R)
        BMLAS - flight quality body mounted linear accelerometer
                sensors (Ax,Ay,Az)
        ATITGS- platform INS attitude outputs (phi,theta,psi)
        GETMLS- microwave landing system sensor (azimuth,elevation,
                range)
        RSIMUS- redundant strapdown IMU sensor (RSDIMU). This routine
                is an executive routine for the RSDIMU.
        IRATG1- initialization for the RSDIMU rate gyro module
        ILNAC1- initialization for the RSDIMU linear accelerometer
                module
        ILNAV1- initialization for the RSDIMU navigator module
        LINAC1- RSDIMU linear accelerometer module
        LLNAV1- RSDIMU navigation module
        RATEG1- RSDIMU rate gyro module

name: FIO.FOR
cont: This file contains routines used to save simulation data
        in special formats on disk files, along with routines to
        help perform this function.
subr: SAVIT - saves FINDS time history data in the (binary) PLT
                file in a sequential, run-time user selectable fashion
        PRNTIC- print the run's initial conditions - in special
                table form - on any ASCII disk file
        FSCHED- determine, for a particular sensor, the time, type,
                and level of failure if simulated
        CHKFL - check if a sensor is scheduled to fail in the run,
                and return the time and type of the scheduled failure
        FLEVEL- determine the failure level of a scheduled failure
        OUTDAT- print out a one line message followed by a formatted

printout of a scaled vector (scaled by a supplied
conversion factor)

TLOUT - print an "event" in a special coded form (described in
section 4.2 of [1]) in the time line (.TLN) file

name: FUTSUB.FOR
cont: This file contains a collection of "utility" routines
       which are generally specific to the FINDS program
subr: ABSLIM- absolute limit - i.e. two-sided limit about zero
       ACCVEL- compute G-frame velocity and acceleration terms
       ROTATV- rotate inertial pos. and vel. vector to the E-frame
       ROTMAT- computes various frame transformation matrices
       RUNGK3- performs Runge-Kutta integration
       RUNWAY- computes A/C position and velocity vectors
               relative to G-frame
       SETUM - sets a scalar into all elements of a vector
       VECM - multiplies two vectors - element by element
       VECS - multiplies two vectors - element by element and
               increments the first
       VECSUM- adds to vectors
       MATV3 - multiplys a 3 by 3 matrix times a vector
       MATTV3- multiplys the transpose of a [3 x 3] matrix
               times a vector
       MATMUL- multiplies a matrix times a vector
       MOVUM - equates two arrays
       DGATIO- prints out a double precision matrix
       SUMMER- computes the conditional average sum of an array
       ASUMER- computes the conditional average sum of an array
       MAXMIN- locates the maximum and minimum elements
               in an array
       MAXMINS-same as MAXMIN - except single precision version
       MXMN2 - same as MAXMIN - except elements are conditioned
               on a non-zero element in a second array
       VECHG1- collapse or expand a vector
       MATCG2- collapse or expand a matrix
       IMTCG2- adds or deletes rows (columns) of matrices
       PNTINV- pointer vector inverse
       LIMVAL- vector limiter for symmetric limits about zero
       LIMVL2- limiter for anti-symmetric two-sided limits
       NOISEG- generates samples from a normal distribution

15

                      with zero mean and unity variance

BARN1 - genertes samples from either a gaussian or a uniform distribution

GAUSS - gaussian random number generator

UNIFRM- uniform random number generator

NAMFIL- forms a file name with a fixed name and various extensions

name: FVMSUB.FOR

cont: This file contains routines which perform operations on vectors and matrices. Unless explicitly stated, all routines operate on double presision quantities.

subr: BUBBLE- perform bubble sort on an array of integers

DOT - compute dot (or inner) product between two column vectors

DOT2 - computes dot product of two row vectors

DOT3 - computes dot product between a row and a column vector

VADD - increments a given vector by a second vector

VADD1 - increments a given row vector by a second row vector

VSCALE- sets one vector equal to another times a scale factor

SEQNCE- initializes an integer array as [1,2,...N]

INSRTN- maintains a pointer vector (integer) with unique entries

VECNULS-initializes a column vector to zero (single precision)

VECNUL- initializes a column vector to zero (Double precision)

SWAP - Swaps a row, column, or diagonal between two matrices

VMAT1 - multiplies a vector by a matrix Y=AX

VMAT2 - computes the vector matrix product sum Y=Z+AX

GMINV - computes the inverse or generalized Penrose inverse of a matrix

MMUL - computes the matrix product Z=XY (with sparseness test on X)

MMUL2 - computes the matrix product Z=XY (with sparseness test on Y)

MAT1 - computes the matrix product Z=XY

16

MAT1A - computes the matrix product Z=XY (Z can equal Y)

MAT2 -   computes the matrix product Z=XY' (for Z
         symmetric)

MAT3 -   computes the matrix product Z=XYX'

MAT3A - computes the matrix product Z=X'YX

MAT4 -   computes the matrix product Z=X'Y

MAT5 -   computes the matrix product Z=XY' (with
         sparseness test on Y)

MAT6 -   computes the matrix product Z=XY' (with
         sparseness test on Y, and Z symmetric)

MADD1 - adds two matrices

MADDI - adds a scaled matrix plus a scaled identity matrix

EQUATE- equates one matrix to another

MATNUL- initializes a matrix to zero

MSCALE- scales a matrix by a scalar constant

TRANS1- computes the transpose of a matrix

17

3.3 Detailed Descriptions Of FINDS Routines

3.3.1 Documentation For File: FMAIN.FOR –

name: FINDS – (Main Program)
        Detection System"

func: This program unit is responsible for coordinating the run-time
        operation of the program. The overall purpose and use of the
        program – from a users point of view – is documented in detail
        in [1]. To show the overall scope and flow of the program a
        functional flow diagram is shown in Figure 2. Three stages of
        the program are evident in this figure:
        * an initialization stage – designed to initialize all
           variables and routines and to establish all disk file
           interactions
        * a basic simulation loop – whose purpose is to continually
           compute the current control signal over the next simulation
           interval, integrate the A/C equations of motion, simulate
           the A/C and sensor subsystems, and exercise the FINDS FDI
           and estimation algorithms until a stopping criteria has
           been satisfied.
        * a termination stage – once the simulation loop has
           satisfied its stopping criterion, the program is gracefully
           terminated.
        Figure 3 provides a much more detailed and annotated flow
        diagram which clearly shows how program FINDS operates.

ref: ACCVEL, ACEQIN, AIRSPS, ALTYP, ATITGS, AUTLD, AUTLDI, AUTTHR,
        BANKTR, BMLAS, BMRGS, BREEZE, CNTRLS, CTEXT, GETMLS, GROUNE,
        INITAL, ISPEC, MATMUL, NAMFIL, NAV, OPN2, PAGEFD, PRNTIC,
        RADALS, ROTATV, ROTMAT, RSIMUS, RUDDER, RUNGK3, RUNWAY, SAVIT,
        SERVO, SET, STABCN, THRUSD, TLOUT, WTHDR1
            Also from the VMS libraries:
        ASIN, CLOSE, DATAN2, DCOS, DSIN, LIB$FLT_UNDER,
        LIB$INIT_TIMER, LIB$STAT_TIMER, OPEN, SECNDS

comm: ALPCOM, ANGLES, ANGS, ARSTAT, ATMO, AZELRN, COEFGE, CONTRL,
        CPU, CRTE, DROP, EARTH, EGUIDE, EKF1, FCOM1, FCOM2, FILNAM,
        FLTCTL, FTITL1, GEARLD, GSLOPE, GUIDE, HICOM, ICLALO, IEST,
        IMLS, INOU, IUVW, LAND, LAOUT, LOGIC4, MCONCO, MLSALL, MLXYZ,

18

NAVINF, NWPLT1, PHILLY, PLOTS, PORT, PORN, PSIR, QRAND,
RGUIDE, RIOUT, RSTATE, RUNGEK, SETCOM, SIGTAU, SIMCOM,
SNSIDT, SNSRDT, SPCFOR, START, SYNC, TRANS, TURN, TURNOF,
UPDAT, VARLAT, VARLON, VORTAC, WIND, WINDCO, WP, XOYOZO

Figure 2. Functional Flow Diagram for Program FINDS

20

```
┌──────────┐
│  START   │
└──────────┘
     │
     ▼
╔════════════╗
║    SET     ║
╚════════════╝
     │
     ▼
╔════════════╗
║  INITIAL   ║
╚════════════╝
     │
     ▼
 prompt for filenames
 of input data and
    output files
     │
     ▼
  open all input
  files for reading
     │
     ▼
 open the SUM and TLN
 file for writing
     │
     ▼
 read from namelists
 ACSTAT,WINDAT,PROGCN
    PRNTPL,GSLOC
     │
     ▼
  no  ◇ plots? ◇  yes
```

- FNDK1 = 'NONE'
- IBINAC = 0? (no → open the PLT file as a binary output file; yes → open the PLT file as an ASCII file)
- kprint > 0? (yes → OUTPS = .TRUE.)
- OUTPS? (yes → open the OUT file for writing; no → IPRNT = 0, FNDK6 = 'NONE')

```
┌──────────┐
│    O     │
└──────────┘
```

(Second column, starting from O)

- accept run specific comments from the TTY
- write a header on the TLN file
- accept a random number seed, JRAND, from the TTY and force it to be and odd number
- OUTPS? (yes → PRNTIC(KDSK6,1))
- PRNTIC(KDSK11,1)
- IEST = 1? (yes → filter = .TRUE.)
- OUTPS? (yes → write namelists on OUT file)
- initialize glideslope parameters

```
┌──────────┐
│    1     │
└──────────┘
```

Figure 3. Flow Diagram for Program FINDS

21

1

initialize A/C parameters → ACEQIN

initialize winds computation → BREEZE

Compute initial attitude

trim initialization for stabilizer → STABCNL

Determine initial pos & velocity in inertial & earth frames

Establish initial A/C state & its derivative

Initialize the sensor modules

initialize FINDS fault tolerant navigator → NAV

Close input files

Rotate accelerations to L-frame and compute A/C pos in G-frame

initialize auto land control logic → AUTLDI

initialize bank turn and course waypoint logic → BANKTR

2

2

Initialize plot file ← subroutine SAVIT

INITIALIZATION

top of major simulation loop

3

Iest=0 & IMLS=1?
- yes → Compute G&C signals using fault tolerant navigator quantities
- no → Compute G&C signals using "true" states

BANKTR ← determine RNAV G&C commands

RUNGK3 (AUTLD)

AUTLD

← auto-level control laws

STABCN ← stabilizer trim control logic

RUNGK3 (AUTTHR)

Compute commanded thrust

← throttle control law

4

Figure 3. Flow Diagram for Program FINDS (continued)

22

4

thrust dynamics ┈➤ THRUSD

rudder servo 6
yaw rate damper
dynamics ┈➤ RUDDER

elevator 6
aileron
servo dynamics ┈➤ SERVO

Altitude
to A/C CG
< 120ft? — yes → GROUNE ◄┈ ground effect computation

no

-ZPos < 100ft.
and IDH=0? — yes → IDH = 1 → Print CAT-II decision stats

no

PLOTS? — yes → SAVIT ◄┈ record variables in the PLT file

no

Time to
print? — yes → Print variables in the out file

no

5

Figure 3. Flow Diagram for Program FINDS (continued)

23

Figure 3. Flow Diagram for Program FINDS (continued)

Figure 3. Flow Diagram for Program FINDS (continued)

Figure 3. Flow Diagram for Program FINDS (concluded)

name: INITAL
func: To initialize several program variables - mostly related
        to the guidance and control algorithms. Originally, (in
        program FILCOMP) INITAL was intended to initialize case
        independent quantities - however, since FINDS doesn't allow
        multiple cases in the same physical run, no such distinction
        is made in FINDS.
call: Call INITAL
args: None
refs: None
refby: FINDS
comm: COEFGE, CONTRL, LOGIC4, SYNC, WIND

name: SET
func: To initialize various constants (such as conversion factors)
        and program flags used by FINDS. Originally, (in program
        FILCOMP) SET was intended to intialize case dependent
        quantities - however, since FINDS doesn't allow multiple
        cases in the same physical run, no such distinction is made
        in FINDS.
call: Call SET
args: None
refs: None
refby: FINDS
comm: ALPCOM, ANGLES, ATMO, CONTRL, EARTH, FCOM1, FCOM2, GEARLD,
        HICOM, IEST, MCONCO, NAVINF, NWPLT1, PHILLY, PLOTS, SETCOM,
        SYNC, VARLAT, VARLON, WIND, WINDCO

3.3.2   Documentation For File: FSFDI.FOR -

name: NAV (fault tolerant navigator)
func: This subroutine is an executive program which implements a
        fault tolerant navigator using the FINDS approach. It is
        responsible for initialization, synchronization, and
        execution of all the modules comprising the FTN and FDIR
        logic. Figure 4 shows a detailed flow diagram indicating
        its operation.
call: Call NAV (Iabort)
args: Iabort - integer   out   flag indicating whether to
                continue or abort the run. If Iabort=0 continue the
                run; otherwise abort.
refs: CHKRAD, CKUNST, DECIDE, DETECT, FILCOL, GTOI, HEALR, INITG,
        KALMN, LIB$INIT_TIMER, LIB$STAT_TIMER, PRNTIC, RECONF, RESCMP
        SUMIN, SUMOUT, TLOUT, WAYPNT
refby: FINDS
comm: CMPSTF, CNTROL, CPU, DCIDEI, DETINF, EARTH, EKBFO, EKF1,
        FCOM1, FILTRT, FLTCTL, GBLEND, IMLS, INOU, MAIN1, MAIN2,
        PHILLY, PLOTS, SIMCOM, SYSU1, SYSXBO, SYSYBO, SYSXW1, SYSX1,
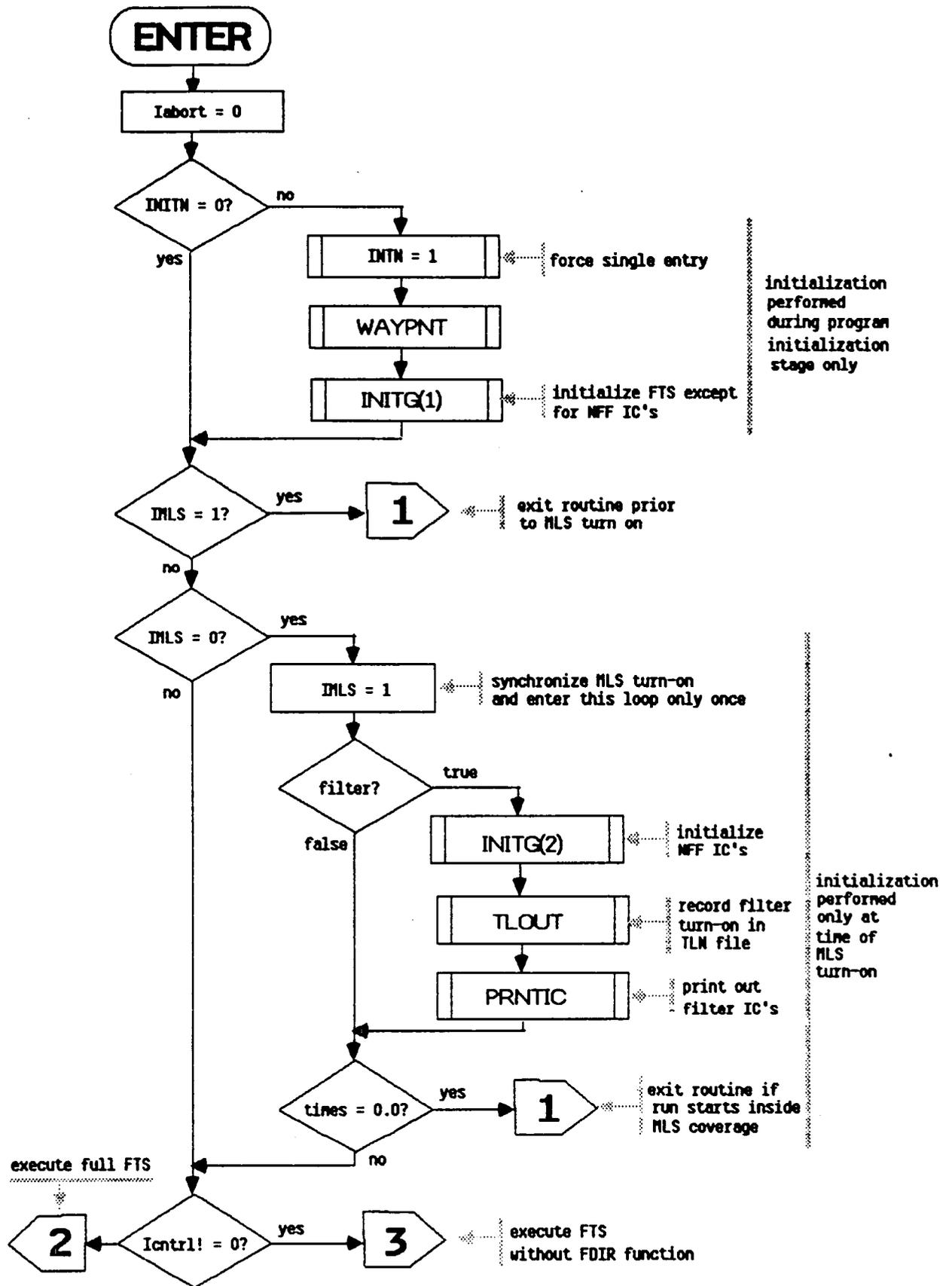
Figure 4. Flow Diagram for Subroutine NAV
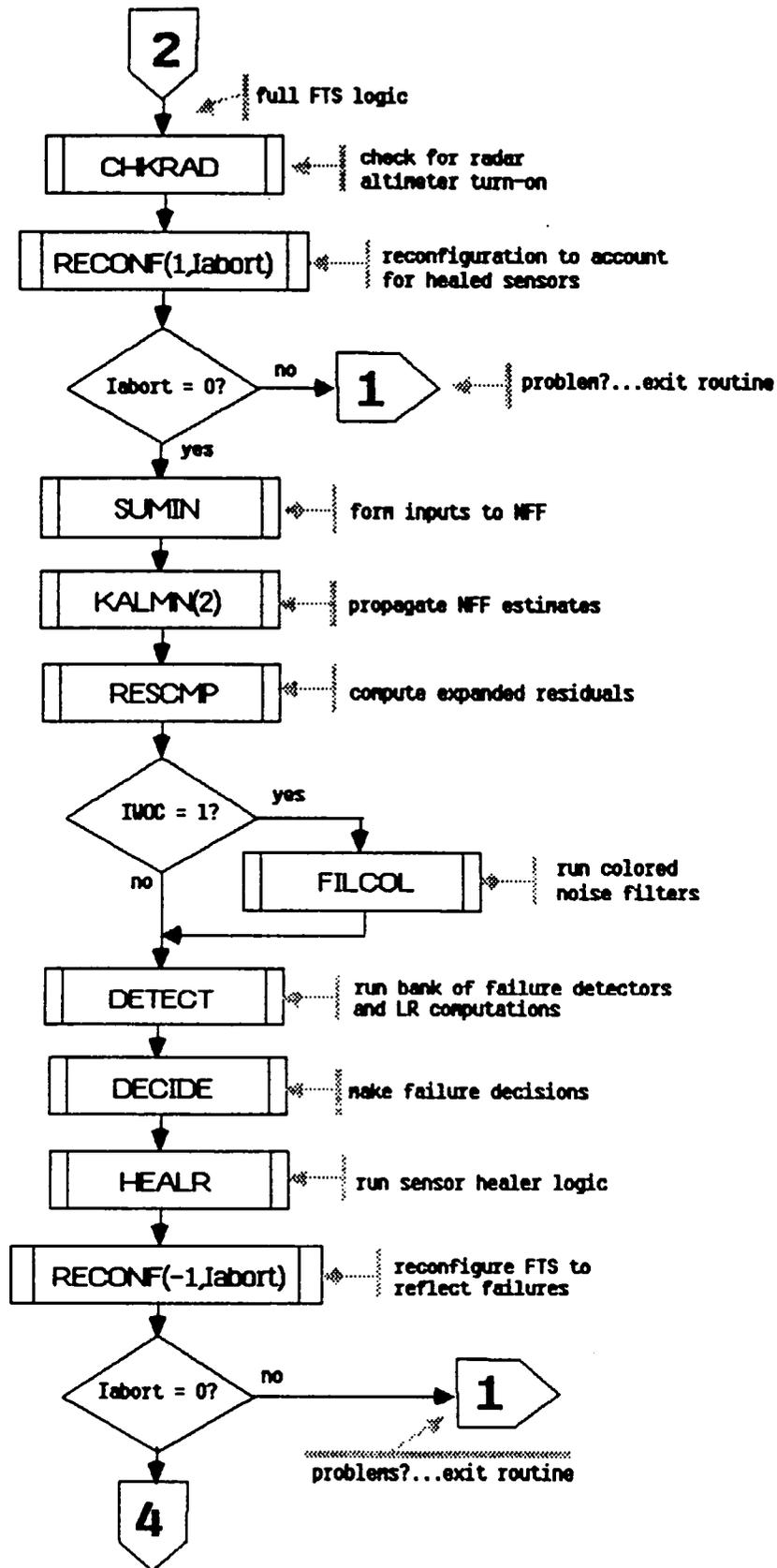
Figure 4. Flow Diagram for Subroutine NAV (continued)
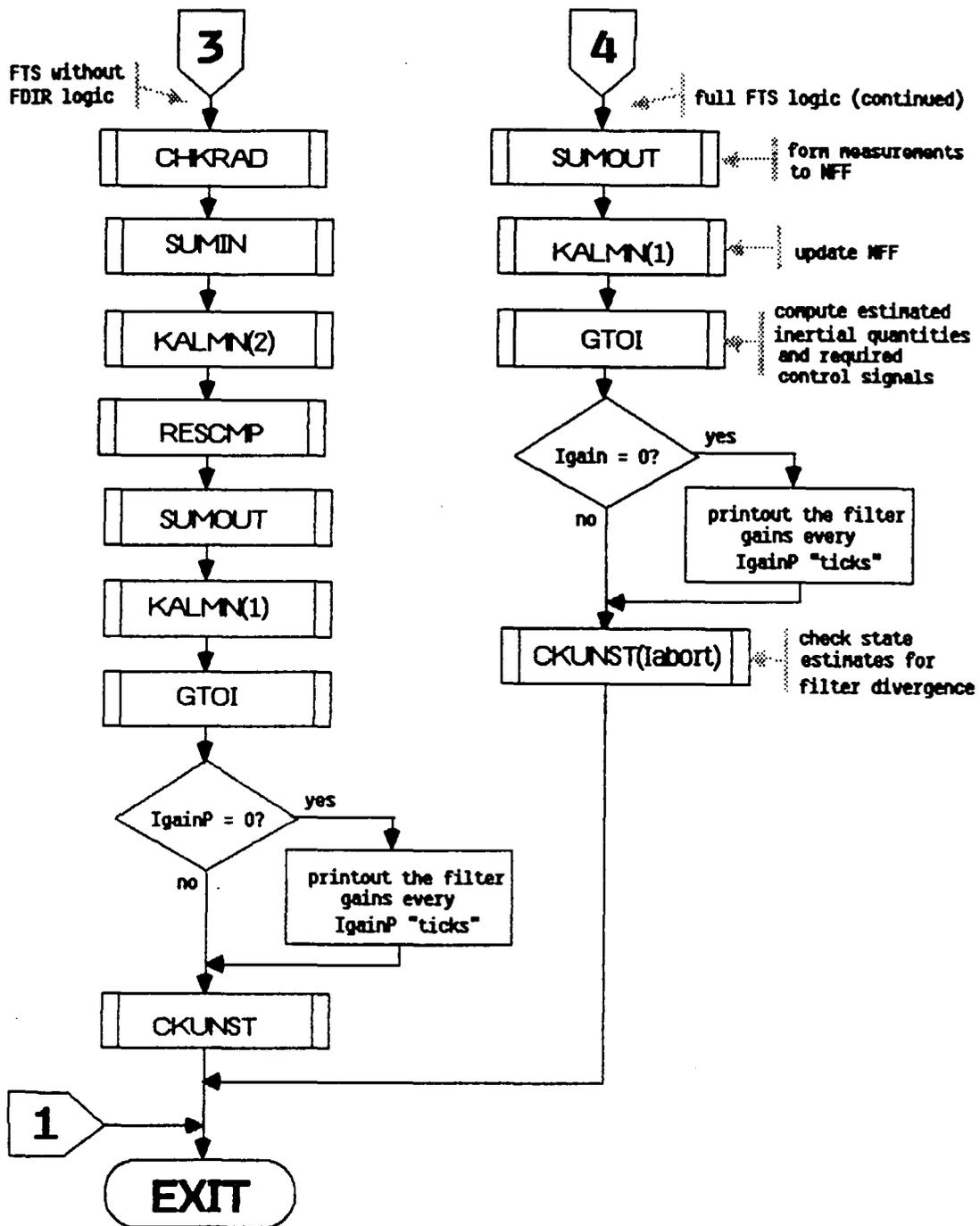
Figure 4. Flow Diagram for Subroutine NAV (concluded)

31

name: INITG

func: Performs initialization for the FTS software. In particular,
         the no-fail filter, failure detection isolation and
         reconfiguration modules are intialized here. In addition,
         program flags and data structures are intialized which
         determine the structure of the FTS. These flags are
         modifed via namelist FILTIN, which is read in this routine.

call: Call INITG (Ipart)

args: Ipart - integer in   flag to indicate which part of the
                 intialization is to be performed. If Ipart=1 all
                 initialization except for determining the initial
                 conditions of the NFF is performed. Otherwise the
                 NFF IC's are chosen

refs: ALTYP, BUBBLE, CONVRF, EQUATE, GTOI, IMSCLE, INITF, MTH$DLOG
         SEQUNCE, STARTF, SUMIN, SUMOUT, UPDA, UPDB

refby: NAV

comm: AGMP, ARMP, ARSTAT, ASMP, CMPSTF, CNEST, CNTROL, COLFIL,
         DCIDEI, DETINF, DETSIG, DETXBI, DETYBI, EARTH, EKBFO, EKF1,
         FCOM1, FILTIC, FILTRT, FLTCTL, FTITL1, GBLEND, GRMP, HEALCM,
         HFCOM, IMLS, INITVL, INOU, LAMP, LOGCI4, MAIN1, MAIN2, MCONCO,
         MLSALL, MLSMP, MULTDT, NWPLT1, PLOTS, PSIR, RALMP, RGMP,
RIOUT,

         SENSCM, SETCOM, SIGTAU, SIMCOM, SYNC, SYSU1, SYSX1, SYSXBO,
         SYSYBO, SYSYW1, VARLON, WIND, XOYOZO, YOBSRV,

name: INITF

func: To initialize the EKF's measurement and process noise
covariance
         matrices, RF1 and QF1 respectively, and the measurement
         normalization scaling vector Yscale. The quantities are set
         as follows:
         a) process noises:
            * if using the "standard" sensor set (i.e. irsdf!=0)
                 $QF1(i) = sig(i)^{**}2$ for I=1,..8
            * or if using the RSDIMU (i.e. irsdf=0);
                 $QF1(i) = sig(17)^{**}2$ for i=1,...3
                 $QF1(i) = sig(18)^{**}2$ for i=4,...6
                 $QF1(i) = sig(i)^{**}2$ for i=7,8
         b) measurement noises:
                 $RF1(i) = sig(i+8)^{**}2/n$ for i=1,...8

32

where n=Ireplf (i+nul) i.e., the number of replications of
a particular sensor type currently in use by the EKF.
c) scaling vector:
* if IYSC=0 then Yscale(i)=1.0   for i=1,...8
(i.e. scaling is disabled)
* otherwise
Yscale(i) = 1.0/SQRT[RF1(i)]   for i=1,...8

call: Call INITF
args: None
refs: None
refby: INITG
comm: DETXBI, FILTRT, FLTCTL, SIGTAU, SYSU1, SYSYW1, YOBSRV

name: STARTF
func: To initialize the no-fail filter's state estimates and
        initial error covariance. This is accomplished as follows:
        choose the initial estimation error from a random
        distribution, s.t.
        1) XICerr(i) = SDXic(i)*s   for i=1,NX
            where s is a sample from a normal distribution with
            mean=0 and variance=1, and SDXic is a vector of expected
            standard deviations
        2) set XF1(i) = Xt-XICerr(i)   for i=1,NX
            where Xt represents the "true" or simulated value of
            XF1(i)
        3) initialize the bias-free filter covariance, PF1, and
            the total no-fail filter (bias & bias-free) filter
            covariance, PXF1, to be diagonal matrices with diagonal
            elements:
                PF1(i,i) = PXF1(i,i) = SDPic(i)**2
            where SDPic is a vector of standard deviations for the
            initial filter covariance
        Note: SDXic and SDPic are in user units, therefore this
              routine also performs conversion to program units
call: Call STARTF
args: None
refs: NOISEG
refby: INITG
comm: ANGLES, AZELRN, CMPSTF, EKF1, FILTIC, MAIN1, MCONCO, PSIR,
        ORAND, SYSX1, UPDAT, VARLON, WIND

33

name: SUMIN

func: To provide a proper set of inputs to the no-fail filter.
The input vector presented to the no-fail filter is formed
by SUMIN as follows:

      1)   each group of like replicated input sensors is broken
down into 3 classes; available & used by the filter;
available, but in standby; and failed. SUMIN further
restricts only a single replication to be active, with
all others placed either in standby or detected as failed.

      2)   rate gyro measurements are compensated for earth and
platform rates

      3)   the input vector, UF1, is formed such that trapezoidal
integration will be performed by the no-fail filter
(i.e. $U(k) = 0.5*[u(k)+u(k-1)]$)

      4)   the gravity vector is computed and added to the end of
UF1 such that UF1 is composed of:
$$UF1 = [Ax,Ay,Az,P,Q,R,Gx,Gy,Gz]'$$
where (Gx,Gy,Gz) is the gravity vector expressed in the
G-frame

      5)   if any input biases are being estimated, their current
estimates are subtracted from the NFF input measurements,
UF1

call: Call SUMIN

args: None

refs: GYROCR, SUMMER, VMPRT

refby: INITG, NAV, RECONF

comm: EKBFO, FILTRT, FLTCTL, GRVTYC, LAOUT, MAIN1, MCONCO, RGOUT,
RIOUT, SYNC, SYSU1, SYSXBO

name: SUMOUT

func: SUMOUT forms a set of average measurements, YF1; to be used
by the no-fail filter. It functions as follows:

      1)   each group of like replicated sensors is classified into
two sets; available and to be used by the filter; and
unavailable, failed, or selected out

      2)   each element of YF1 is averaged as:
$$YF1(*) = (1/nr)*[m(1)+m(2)+..m(nr)]$$
where nr is the number of available, replicated
measurement sensors, and m is an arbitrary measurement

      3)   psi measurements are compensated for any runway yaw by:

```
              YF1(7) = YF1(7) - PSIRU
              where PSIRU is the runway yaw to north expressed in
              radians
          4)  if IYSCL.NE.O then each measurement is normalized by the
              expected variance of that signal, s.t.
                  YF1(i) = YF1(i)*Yscale(i)
              where Xscale is set in subroutine INITF
```

call: Call SUMOUT
args: None
refs: SUMMER
refby: INITG. NAV
comm: AGOUT. ASOUT. DETXBI, FILTRT. FLTCTL. MLOUT. PSIR, RAOUT,
          RIOUT. SYSYW1. YOBSRV

name: GYROCR
func: GYROCR computes the correction terms required to compensate
          the rate gyros for earth and platform rates. GYROCR functions
          as follows:
          1) to ensure that gyro measurements are compensated only
              once per simulation "tick", a local copy of the last
              time (TimesL) is saved. If Times <= TimesL then
                  WCOMP(i) = 0.0 for i=1,...3
          2) otherwise:
                  WCOMP = Trb' Trl Wl
              where Trb' is the transformation from the runway to the
              body frame of reference and Trl is the transformation
              from the local level to runway frame. Wl is the frame
              rates expressed in the local level frame
          Note: most of the variables used in this subroutine are
                  computated in GTOI.
call: Call GYROCR (wcomp)
args: wcomp - double out   vector of compensation terms to be
                  subtracted from the rate gyro measurements (see
                  description above.)
refs: MATTV3
refby: SUMIN
comm: ARSTAT. EARTH. PSIR, SIMCOM. TRBER

35

name: GTOI
func: GTOI forms estimates for inertial position, velocity, and
          acceleration, and runway acceleration. It also computes
          the A/C's current longitude and latitude along with their
          rates of change. In addition, Tic, the last column of Tlc,
          coriolis and centripetal correction terms for compensating
          the platform gravity force, and several control variables
          required by the G&C logic are also all computed.
call: Call GTOI
args: None
refs: ASUMER, MATV3, MTH$DATAN2, MTH$DCOS, MTH$DSIN, MTH$DSQRT
refby: INITG, NAV
comm: ARSTAT, EARTH, EGUIDE, EKF1, FILTRT, GRVTYC, IMLS, MAIN1,
          MCONCO, PSIR, RGOUT, SYSU1, TRBER,

name: CKUNST
func: CKUNST checks the no-fail filter estimates for divergence
          and sets an abort flag (Iabort) if a divergence criteria is
          exceeded. The primary benefit of this routine is to reduce
          computation time (and associated costs) by ending a divergent
          run early. The following divergence criteria is used, where
          divergence is declared if:
          1) the altitude estimate is below the runway i.e. $XF1(3)<0.0$
          2) the absolute sum of the position errors are greater than
              a position error bound, POSBND, i.e.
                  $sum\{|posit(i)-XF1(i)|\}>POSBND$ for i=1,..3
          3) the absolute sum of the velocity errors exceeds a velocity
              bound, VELBND, i.e.
                  $sum\{|VELOC(i)-XF1(3+i)|\}>VELBND$ for i=1,...3
          4) or the absolute sum of the attitude errors are greater
              than an angular bound, ANGBND, i.e.
                  $sum\{|a(i)-XF1(6+i)|\}>ANGBND$ for i=1,...3
              where a = [Phi, theta, Psi-Psiru]
          If the divergence criteria is met, the stopping time for the
          run, Tstop, is set to the current simulation time, an abort
          flag is set, and messages are sent to the connected terminal
          and the time line file.
call: Call CKUNST (Iabort)
args: Iabort - integer   inout   run abort flag, where:
                   Iabort=-1

36

indicates run should be aborted, and otherwise run should proceed.

refs: ALTYP, TLOUT
refby: NAV
comm: ANGLES, EKF1, FILTIC, PSIR, SETCOM, SIMCOM, UPDAT

name: KALMN
func: KALMN serves as the executive routine to implement an extended Kalman filter, where the plant equations are:
$$X(k+1) = A\ x(k) + B[X(k)]u(k) + E[X(k)]w(k)$$
and the measurement equation is:
$$y(k) = h[X(k)] + u(k)$$
The filter is realized as a lower order bias-free filter followed by a bias filter and a blender to form the bias corrected state estimates. The reader is referred to [3] and [4] for a more detailed description of the approach. KALMN is meant to be called in two passes: once to perform all the filter propagations, and then again to update the estimates and covariance with the measurements. The following user supplied routines are required to define the non-linear terms:

        UPDA   defines A
        UPDAB defines ABF1
        UPDB   defines B[X(k)]
        UPDO   defines O[X(k)]
        UPDH   defines h[x(k)]
        UPDPH defines HP1

call: Call KALMN (Iup)
args: Iup -   integer in   update/propagate flag, where Iup=1 enables updating, and Iup=2 enables propagation
refs: BIASF, BLEND, EKFN1
refby: NAV, RECONF
comm: SYSXBO

name: EKFN1
func: EKFN1 represents the bias-free filter portion of the no-fail
          filter. It is implemented as an extended Kalman filter.
          Covariance propagation of the stabilized normal equations
          is performed. The state estimates, XF1 are NOT computed in
          this subroutine, rather they are formed in subroutine BLEND.
          To accomodate reconfiguration due to the failure or a healing
          of a sensor, the state and covariance at time k/k is stored
          temporarily in RBF0. Figure 5 details this module. The
          reader is also referred to [3] and [4] for a detailed
          description of the no-fail filter's implementation.
call: Call EKF1 (Iup)
args: Iup -   integer in   update/propagate flag
refs: EQUATE, GMINV, MADD1, MADDI, MAT1A, MAT2, MAT3, MAT5, MMUL,
          MMUL2, MSCALE, UPDAB, UPDB, UPDPH, UPDQ
refby: KALMN
comm: EKF1, FILTRT, FLTCTL, MAIN2, SYSU1, SYSX1, SYSXBO,
          SYSYBO, SYSYW1, TSTORE

Figure 5. Flow Diagram for Subroutine EKFN1

39
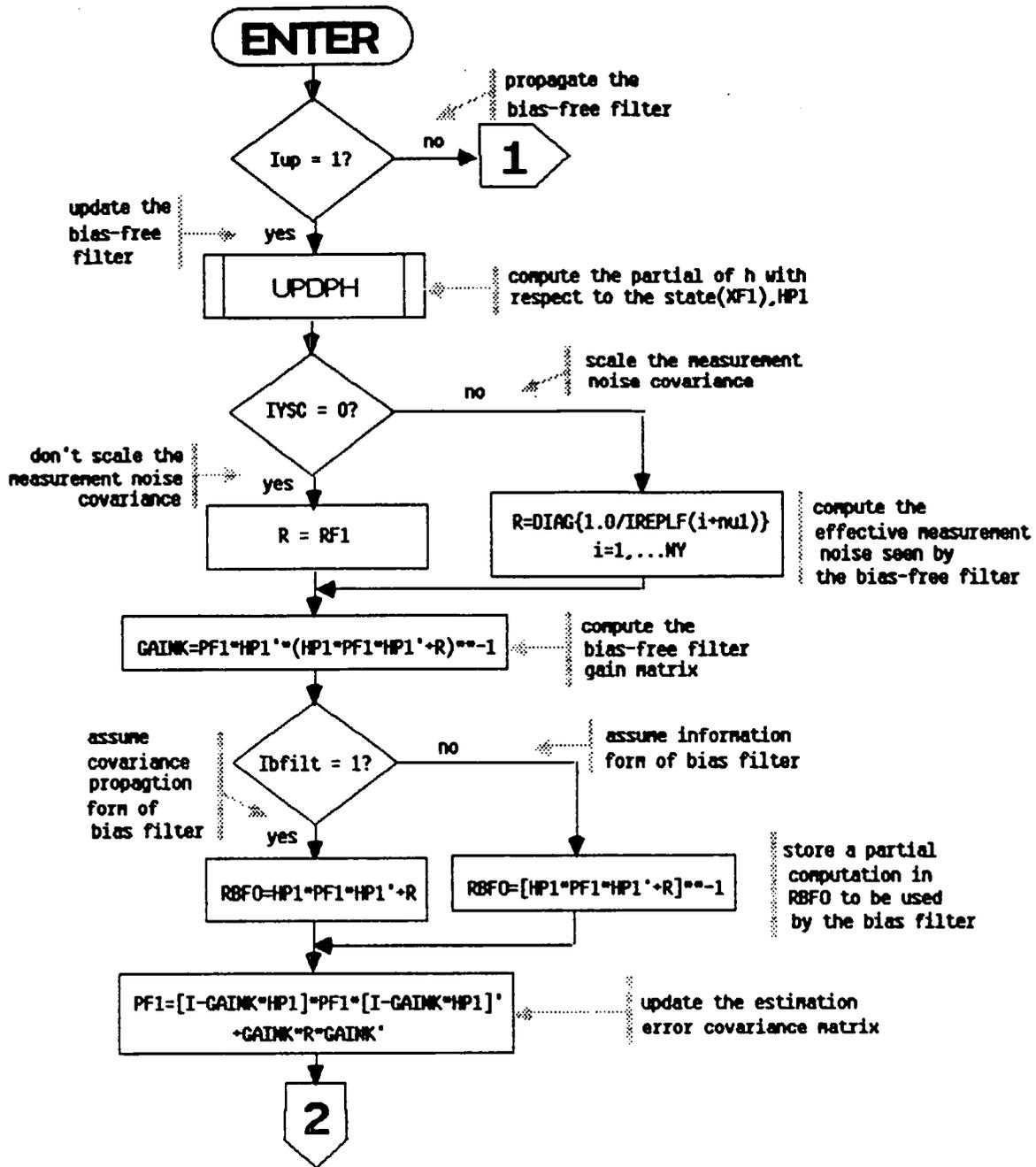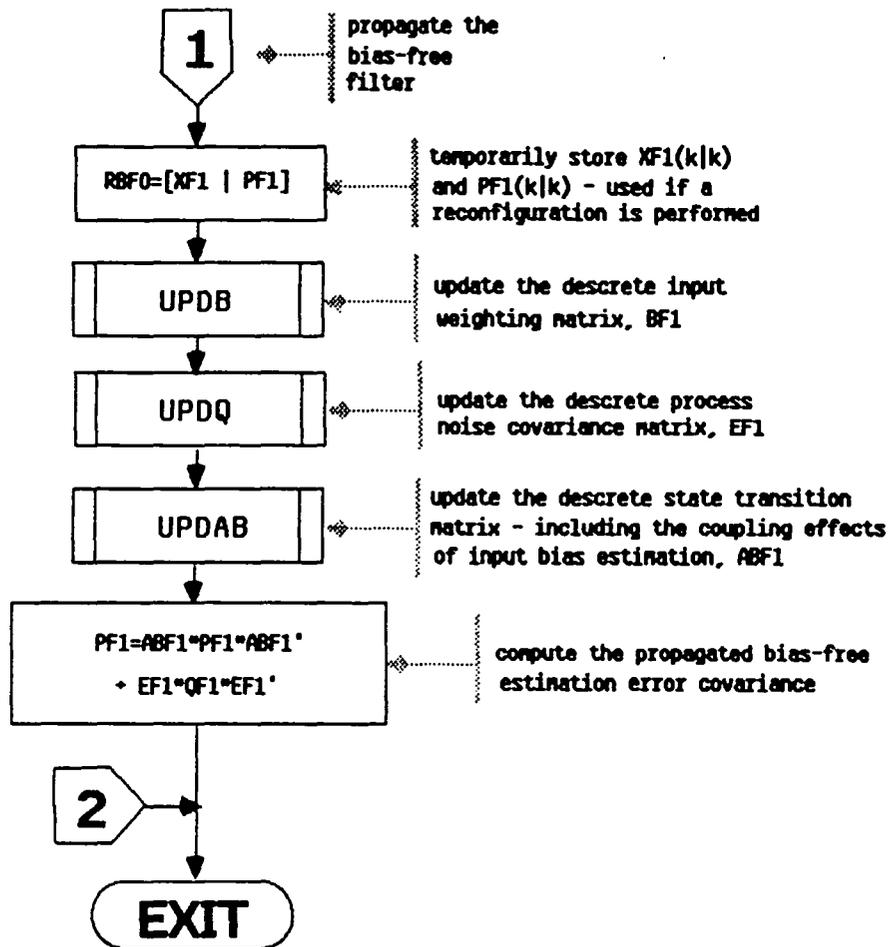
Figure 5. Flow Diagram for Subroutine EKFN1 (concluded)

name: BIASF
func: BIASF implements the bias filter portion of the no-fail
         filter. The operation of this routine is shown in Figure 6.
         The reader is referred to [3] and [4] for detailed
         descriptions of the no-fail filter implementation. A
         software switch exists in this routine which can be set at
         compile time or at run time via the Fortran debugger. The
         switch is IGNC: if IGNC=1 use an anti-svmetric equation for
         PBFO, otherwise use a (more complicated) symetric equation.
call: Call BIASF (Iup)
args: Iup -   integer in   update/propagate flag
refs: ALTYP, BLGAIN, DGATIO, EQUATE, GMINV, MADD1, MADDI, MAT1,
         MAT1A, MAT3, MAT3A, MAT4, MMUL, VMPRT
refby: KALMN
comm: CMPSTF, EKBFO, EKF1, FILTRT, FLTCTL, GBLEND, INOU, MAIN1,
         MAIN2, SYSU1, SYSX1, SYSXBO, SYSYBO, SYSYW1, TSTORE, YOBSRV

41

ENTER

no propagation step required - since
there are no bias filter dynamics

Iup = 1? →no→ 2

perform bias filter
measurement update   yes

Note: BFlub is not a
variable- rather it
represents the matrix
of columns of BF1
corresponding to inputs
with estimated biases

B = [-BFlub|0]
D = [0|I]
CBFO=HP1*[ABF1*VBO+B]+D

compute the effective bias filter
measurement matrix CBFO, where the
partial term CBFO=[ABF1*VBO+B] was
saved previously in subroutine BLEND

BLGAIN(VBO)

update the no-fail
filter blender gain, VBO

computations appropriate for
Information form of bias filter

IBfilt = 1? →no→ 1

computations for
covariance propagation
form of bias filter   yes

compute bias filter
gain matrix
Note: RBFO=HP1*PF1*HP1+R
was saved in subroutine
EKFM1

GAINBO=PBFO*CBFO'*[CBFO*PBFO*CBFO'+ RBFO]**-1

use symmetric equations

IGNC = 1?   no

use anit-symmetric
equations   yes

PBFO=[I-GAINBO*CBFO]*PBFO

TMP1 = [I-GAINBO*CBFO]
PBFO=TMP1*PBFO*TMP1'
+ GAINBO*RBFO*GAINBO'

update bias filter
estimation error
covariance matrix

2   return

Figure 6. Flow Diagram for Subroutine BIASF

42

1

computations for information form of the bias filter

PBFO=PBFO + CBFO*RBFO*CBFO

update the bias filter INFORMATION matrix,
PBFO; where PBFO=[HP1*PF1*HP1+R]**-1
was stored previously by subroutine EKFN1

TMP2 = PBFO**-1

invert PBFO to obtain the error covariance
Note: this step is optional and should be
done only if the error covariance is required

TMP2 = full rank?

no

yes

GAINBO=PBFO*[HP1*VBO+D]'*R**-1

GAINBO=0

compute the
bias filter
gain matrix

2

EXIT

Figure 6. Flow Diagram for Subroutine BIASF (concluded)

43

name: BLEND
func: BLEND computes the bias and bias-free state estimates and
          "blends" them together to form the total state and bias
          estimates. It also forms the total state and bias estimation
          error covariance and Kalman gain matrix. Figure 7 details the
          operation of BLEND.
call: Call BLEND (Iup)
args: Iup -   integer in   update/propagate flag
refs: EQUATE, MADD1, MAT1, MAT4, MMUL, TRANS2, UPDH, VECNUL, VMAT1,
          VMAT2
refby: KALMN
comm: CMPSTF, DETINF, EKBFO, EKF1, FILTRT, FLTCTL, GBLEND, MAIN2,
          SYSU1, SYSX1, SYSXB0, SYSYB0, SYSYW1, TSTORE
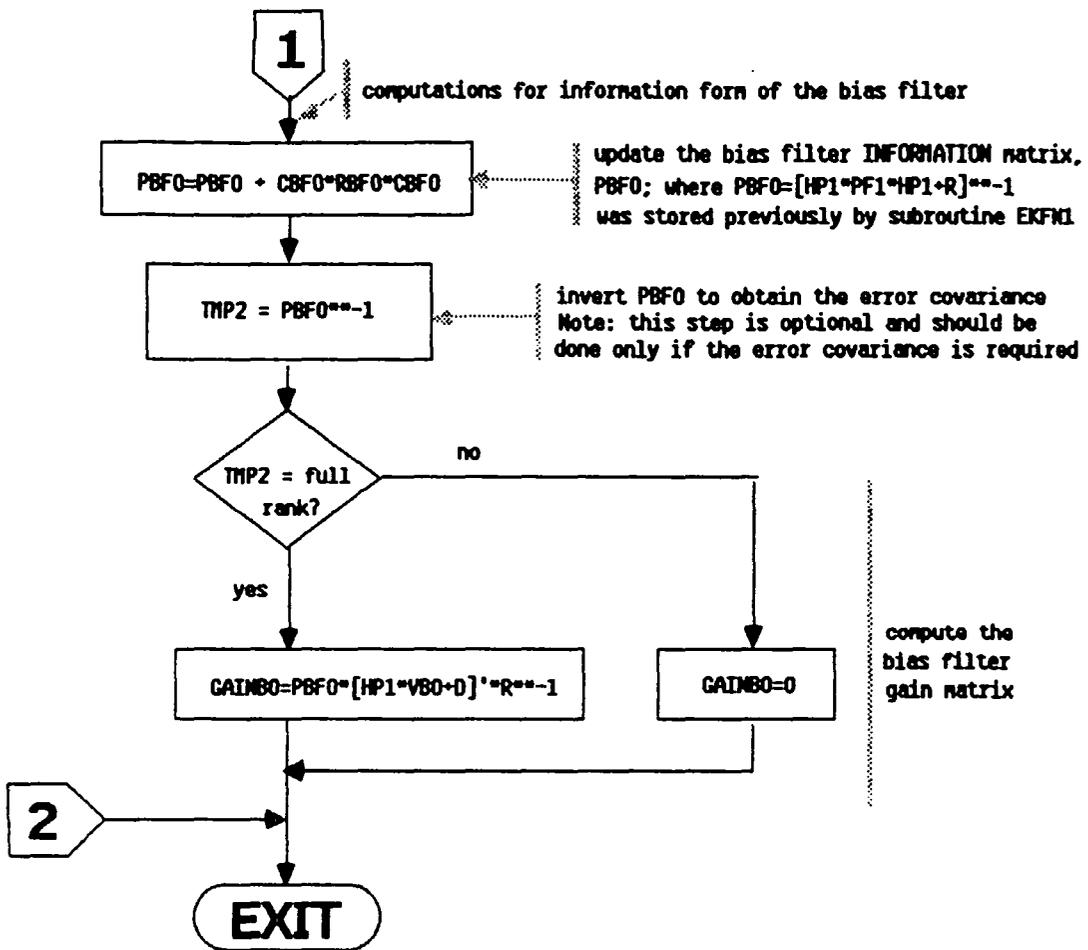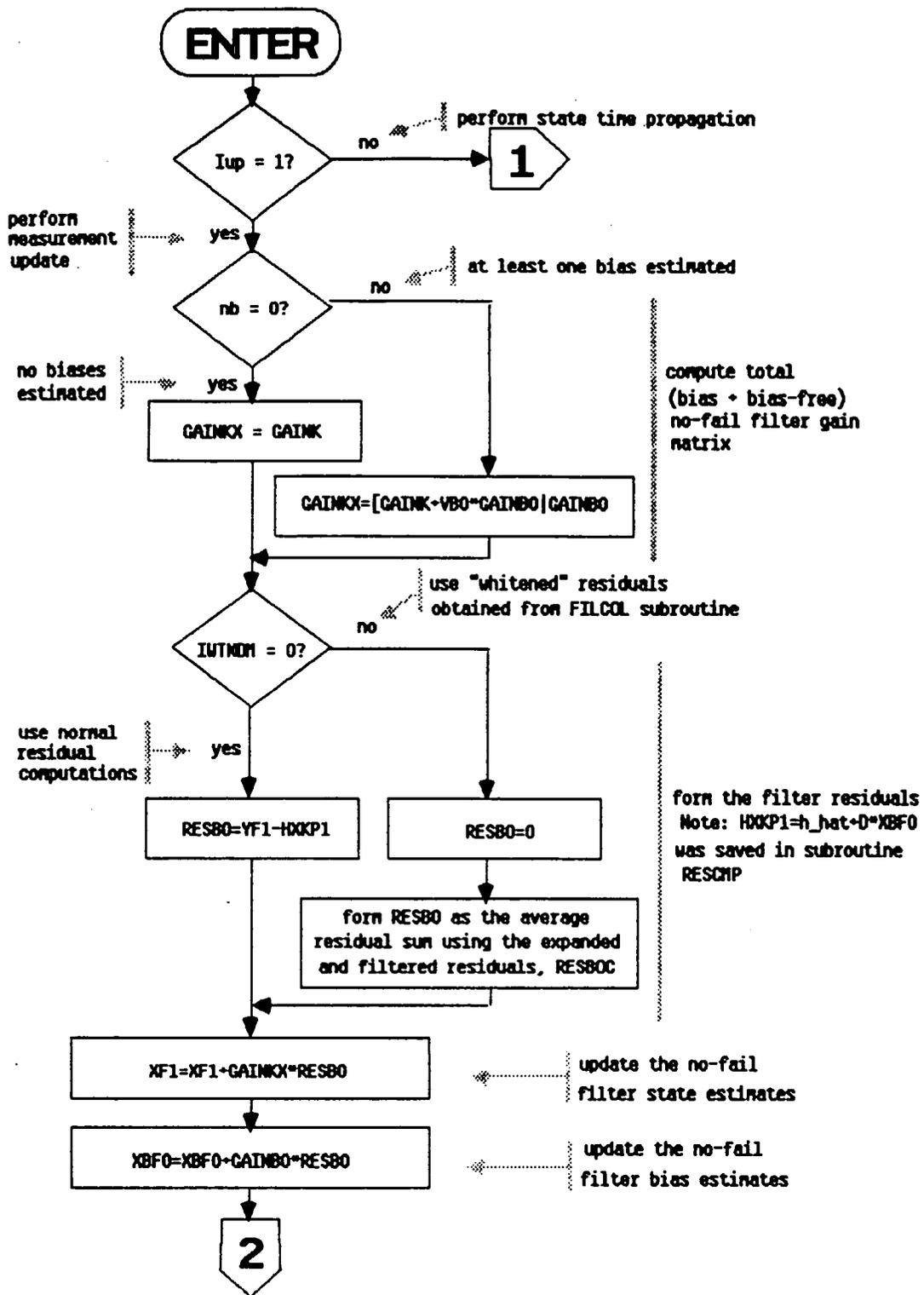
Figure 7. Flow Diagram for Subroutine BLEND
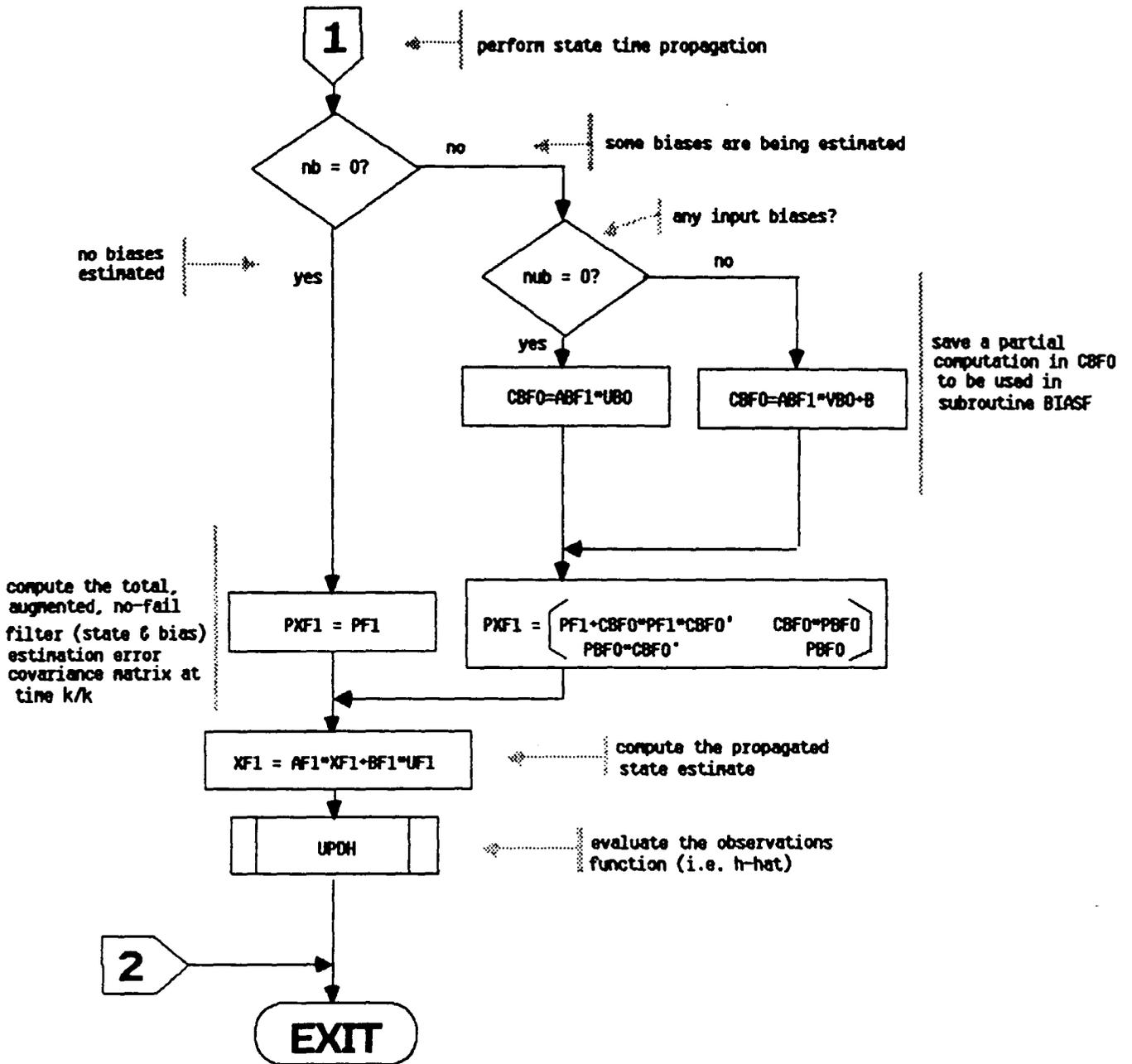
Figure 7. Flow Diagram for Subroutine BLEND (concluded)

46

name: BLGAIN
func: BLGAIN computes the blender gain, VB, in a recursive fashion.
        VB is computed recursively as:
                VB(i+1) = [I-GAINk*HP1]*ABF1*VB(i)
                            + [-BF1ub + GAINk(HP1*BF1ub-D)]
        where the second term is computed as the augmented matrix:
                [(GAINP*HP1-I)*BF1ub , -Gaink*D]
        BF1ub refers to a matrix built out of the columns of BF1.
        where each column, corresponding to biases which are
estimated,
        is included (augmented together) to form BF1ub.
call: Call BLGAIN (VB)
args: VB -   double inout   updated blender gain matrix
refs: MADD1, MADDI, MAT1, MMUL2, MSCALE
refby: BIASF
comm: EKF1, MAIN1, MAIN2, SYSU1, SYSX1, SYSXBO, SYSYW1, TSTORE,
        YOBSRV

name: SETISN
func: SETISN maintains the value of a vector called ICNTSN. The
        ordering of elements in ICNTSN are constant and correspond
        to the absolute replicated sensor ordering found in Table 6.
        The value of each element is the location in UF1 for the
        first six elements, and the location in the expanded inno-
        vations, RF1, for the rest of ICNTSN.
        ICNTSN provides a "mapping" between an absolute indexing
        scheme and a particular (collapsed - due to failures) indexing
        scheme used by the NFF
call: Call SETISN
args: None
refs: IMSCLE
refby: RECONF
comm: DETINF, DETXBI, FILTRT, SYSU1

47

name: CHKRAD
func: CHKRAD checks radar altimeter turn-on criteria. If the
          following criteria are satisified the radar altimeter
          measurements are added to the no-fail filter measurements,
          and the MLS elevation measurements are removed (selected out).
                    RADAR = false (i.e. radar switchover has not occured)
                    and |XF1(3)| < Hradar (A/C is below a fixed altitude)
                    and Irepls(6)!=0 (vertical accelerometers available)
          Radar altimeter measurements are added by "healing" them, and
          performing the reconfiguration required. In addition, if the
          filter covariance is too small for x and x-dot, it is boosted
          to force the radar altimeter measurements to be used by the
          no-fail filter.
call: Call CHKRAD
args: None
refs: RECONF, TLOUT, VECNUL
refby: None
comm: CMPSTF, DCIDEI, DETXBI, EKBFO, EKF1, FILTRT, FLTCTL, HEALCM,
          HFCOM, INOU, LOGIC4, PLOTS, SENSCM, SIMCOM, SYSXBO, SYSYW1

name: UPDA
func: UPDA updates the discrete state transition matrix (AF1).
          Currently AF1 is a constant so UPDA is called only once. AF1
          is defined in equation (2.2.13) on page 29 of [2]
          (where A=AF1).
call: Call UPDA (nr,nc,x,A)
args: nr -   integer in   currently not used
          nc -   integer in   currently not used
          x -   double in   currently not used
          A -   double out   updated discrete state transition
matrix
refs: MTH$DEXP
refby: INITG
comm: MAIN1, SIGTAU, SYNC

name: UPDAB
func: UPDAB updates the discrete state transition matrix (ABF1) to
         include the coupling due to the estimation of input
         measurement biases. It also computes and saves a matrix of
         partials needed for the bank of detectors BDFI.
         Computationally UPDAB computes:
         1) ABF1 = AF1 + partial of BF1*(UF1) w.r.t. XF1
         2) BDFI = partial of BF1**BFIu, w.r.t. phi, theta, psi where
             XBFIu is a vector of failure estimates for input sensors.
         The reader is referred to pages A-1 in [3] for a description
         of the partial derivative terms required for this module.
call: Call UPDAB (ns,nu,u,AB)
args: ns -   integer in   currently not used
         nu -   integer in   currently not used
         u -   double in   input measurement vectors (UF1)
         AB -   double out   updated discrete state transition
                  matrix
                  which includes the coupling due to input measurement
                  biases.
refs: None
refby: EKFN1
comm: DETXBI, EULER, MAIN1, SYNC, TRBER

name: UPDB
func: UPDB updates the discrete input weighting matrix, BF1, and
         also evaluates and saves:
         1) sines and cosines of the estimated euler angles
         2) the transformation from the B to the R frame
         3) the transformation from the R to the E frame
         BF1 is defined in equation (2.2.13) on page 29 of [2]
         where (B=BF1)
call: Call UPDB (nx,nu,x,B)
args: nx -   integer in   currently not used
         nu -   integer in   currently not used
         x -   double in   vector of current state estimates
                  (e.g. XF1)
         B -   double out   updated discrete input weighting
                  matrix
refs: MTH$DCOS, MTH$DSIN, MTH$DTAN
refby: EKFN1, INITG

49

comm: EULER, FLTCTL, MAIN1, SYNC, TRBER

name: UPDO
func: UPDO updates the discrete process noise covariance matrix,
          OF1. UPDO assumes that UPDB has been called recently,
          therefore Trb and Ter are current. OF1 is defined in
          equation (2.2.14) on page 30 of [2] (where Q=QF1). In
          addition, provisions have been made in UPDO to allow for
          the following modifications to OF1:
          1) modeling errors, not accounted for by the plant and
              measurement equations, can be accounted for partially
              by increasing the process noise variance. Therefore, a
              vector of terms, called DIAGO is added to the diagonal of
              OF1. Currently DIAGO is set to zero and can only be
              changed at compile time or via the debugger at run time.
          2) to represent errors due to scale factor and misalignment
              of the rate gyros, the following terms are added to the
              measurement noise variance for rate gyros:

$$Vrg = V + spm* \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} STMP1 \\ STMP2 \\ STMP3 \end{pmatrix} + scaleF* \begin{pmatrix} STMP1 \\ STMP2 \\ STMP3 \end{pmatrix}$$

              where each of these terms are defined in comments in the
              actual code.
call: Call UPDO (nx,ndistb,V,Q)
args: nx -   integer in   total number of states
          ndistb- integer in   currently not used
          V -   double in   vector of measurement noise variance
                    used by the filter
          O -   double out   updated discrete process noise
                    covariance
refs: LIMVAL, MTH$DEXP
refby: EKFN1
comm: ARSTAT, MAIN1, MCONCO, RGMP, SIGTAU, SYNC, TRBER

name: UPDH
func: UPDH updates the non-linear observations function H. H is
        defined in equations (2.2.15)-(2.2.24) on pages 30 and 31 in
        [2] (where h[x(k)] = H).
call: Call UPDH (ny,nx,X,H)
args: ny -   integer in   currently not used
        nx -   integer in   currently not used
        X -   double in   vector of current state estimates
        H -   double out   updated vector of observations
refs: MTH$DASIN, MTH$DSQRT
refby: BLEND, RCOF
comm: FILTRT, MAIN1, MLSALL, XOYOZO, YOBSRV

name: UPDPH
func: UPDPH updates the partial of H w.r.t. XF1, called HP1.
        HP1 is defined on pages A-3 - A-5 in [3].
call: Call UPDPH (nx,X)
args: nx -   integer in   total number of states
        X -   double in   vector of current state estimates
refs: MSCALE, MTH$DSQRT
refby: CLPSIO, EKFN1
comm: CMPSTF, MAIN1, MLSALL, SYSU1, SYSXBO, SYSYW1, XOYOZO, YOBSRV

name: DETECT
func: DETECT implements a bank of detectors and likelihood ratio
        computers. Each detector estimates the level of a bias
        jump failure - hypothesized to start at the beginning of
        an estimation window - by observing the expanded and filtered
        residuals sequence generated by RESCMP and FILCOL. The
        hypothesized failure is assumed to affect no-fail filter
        input measurements or output measurements only. Therefore,
        a single failure cannot directly enter into BOTH an input
        and an output measurement.

        The bank of likelihood ratio computers operate over a decision
        residual window and are designed to compute the log likelihood

of a singleton sensor failure, or a dual simultaneous failure
in MLS sensors.

Subroutine DETECT functions as an executive of this bank of
detector/LR computers. It is responsible for computing all
common terms required to intialize the parallel bank at the
time of estimation and decision window resets, managing the
estimation and decision window mechanisims, and implementing
the parallel computations in a sequential fashion. The reader
is referred to [1]-[3] for detailed descriptions of the method
of operation, and particularly to figure 1 on page 12 in [2]
which gives a functional description. Since
subroutine DETECT is a key subroutine in the FINDS program,
Figure 8, a detailed flow diagram, is supplied to describe
its operation.

call:	Call DETECT
args:	None
refs:	EQUATE, GMINV, LIB$INIT_TIMER, LIB$STAT_TIMER, LKF, LRT, MADD1, MAT1, MAT6, MATNUL, MMUL, MMUL2, MSCALE, VECNUL, VMPRT
refby:	NAV
comm:	CMPSTF, CNEST, COLFIL, CPU, DCIDEI, DETINF, DETXBI, DETYBI, EKBFO, FILTRT, FLTCTL, HEALCM, INITVL, MAIN1, MAIN2, MULTDT, SENSCM, SYSU1, SYSX1, SYSXBO, SYSYW1, TSTORE, YOBSRV

ENTER

Ifrst = 1? — yes →

initialization (Note: Ifrst is set to 1 in a data statement)

no

Ifrst = 0

CNICG = 1.7 ◄···· scale factor for increasing innovations covariance presented to detectors

are we simulating colored noises for the MLS sensor?

IWOC = 1? — yes →

no

FILaz=(1-GNaz)*PHIaz
FILel=(1-GNel)*PHIel
FILrn=(1-GNrn)*PHIrn

Note: GN* terms are initialized in subroutine INITG and PHI* terms are set in subroutine FILCOL

initialize quantities used in compensating for the colored nature of the MLS measurements

AZcor1=CNICG*[(1-PHIaz**2)
+2.0*GNaz*PHIaz**2]/(1-FILaz**2)
AZcor2=CNICG*(1-PHIaz**2)/(1-FILaz**2)

ELcor1=CNICG*[(1-PHIel**2)
+2.0*GNel*PHIel**2]/(1-FILel**2)
ELcor2=CNICG*(1-PHIel**2)/(1-FILel**2)

RNcor1=CNICG*[(1-PHIrn**2)
+2.0*GNrn*PHIrn**2]/(1-FILrn**2)
RNcor2=CNICG*(1-PHIrn**2)/(1-FILrn**2)

initialize quantities common to all detector/LR's

COM2 = expanded HP1 ◄···· expand the observations partials matrix to account for replicated sensors

TMP2 = COM2'*PXF1*COM2

USE colored MLS noise assumptions?

IWOC = 1? — yes → 1

no

2

Figure 8. Flow Diagram for Subroutine DETECT

53

Figure 8. Flow Diagram for Subroutine DETECT (continued)

Figure 8. Flow Diagram for Subroutine DETECT (continued)

4

are we estimating any normal operating biases in the NFF?

nb = 0?

yes

no

are any of these biases input measurement biases?

Nub != 0?

yes

no

i = 1

II = INOBP(i)

II = pointer to input type

COM2[i]c = HP1*BF1[II]c

i = i+1

i > Nub?

no

yes

form -HP1*BFlub where BFlub represents a matrix formed out of the columns of BF1 which correspond to inputs with estimated biases

III = NX1+NUB

are any biases measurement biases?

NYb != 0?

yes

no

COM2[Nub1,Nb]c = -HP1[III,NXB]c

now COM2 = -HP1[III,NXB]c

TMP2 = GAINKX*COM2

are there any input measurement biases

Nub != 0?

yes

no

TMP2 = BFlub-TMP2

TMP2[NX1,NXB]r = I-TMP2[NX1,NXB]r

at this point TMP2 is the augmented matrix:

CBFI = COM2*VBI

CBFI[i]c = 0
i=1,...NFmax

save portion of the detection measurement matrix

$$\begin{bmatrix} [BF1u-GAINKX*(HP1*BF1u+D)] \\ I-GAINB0*(HP1*BF1u+D)] \end{bmatrix}$$

5

Figure 8. Flow Diagram for Subroutine DETECT (continued)

56

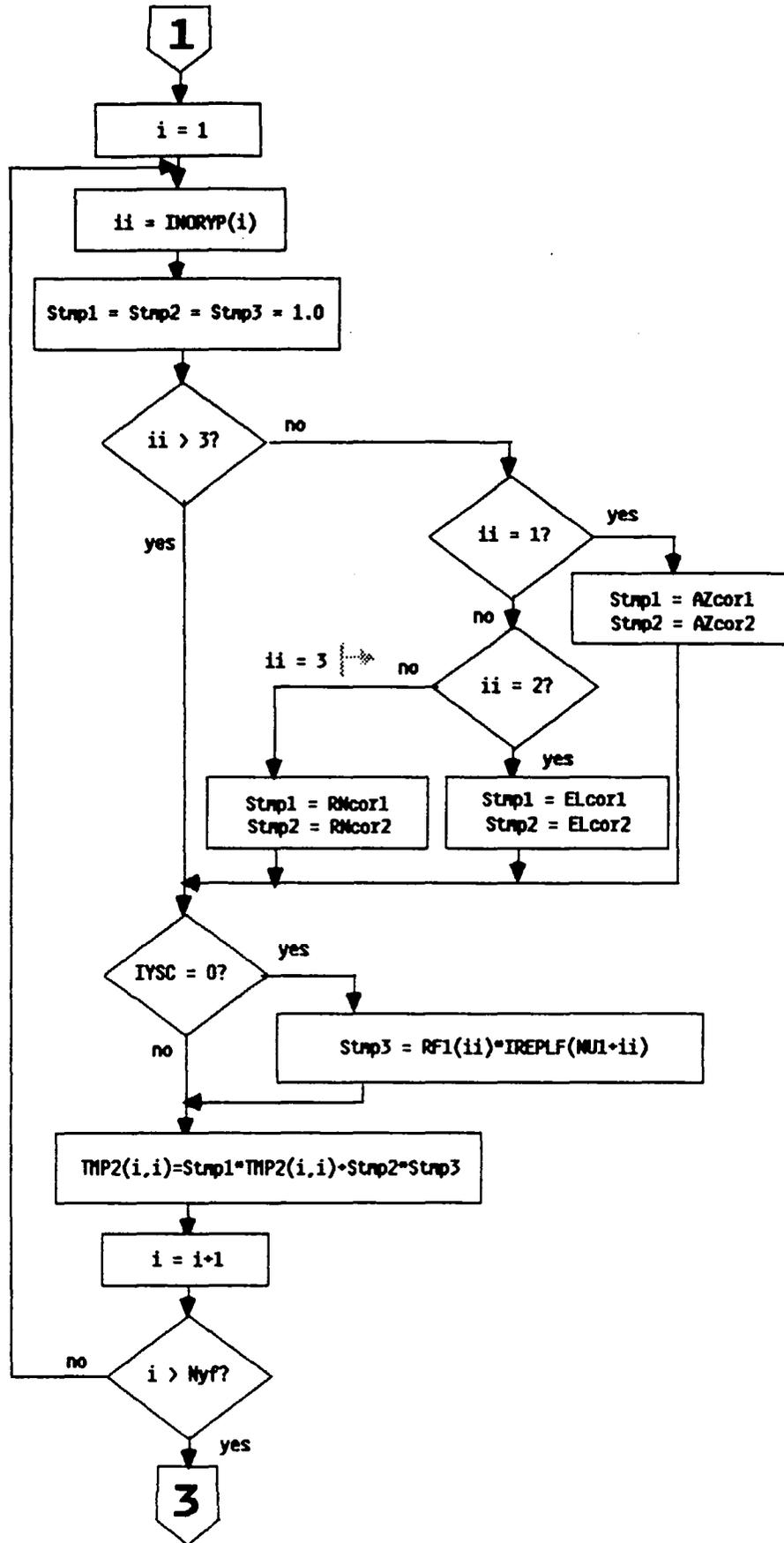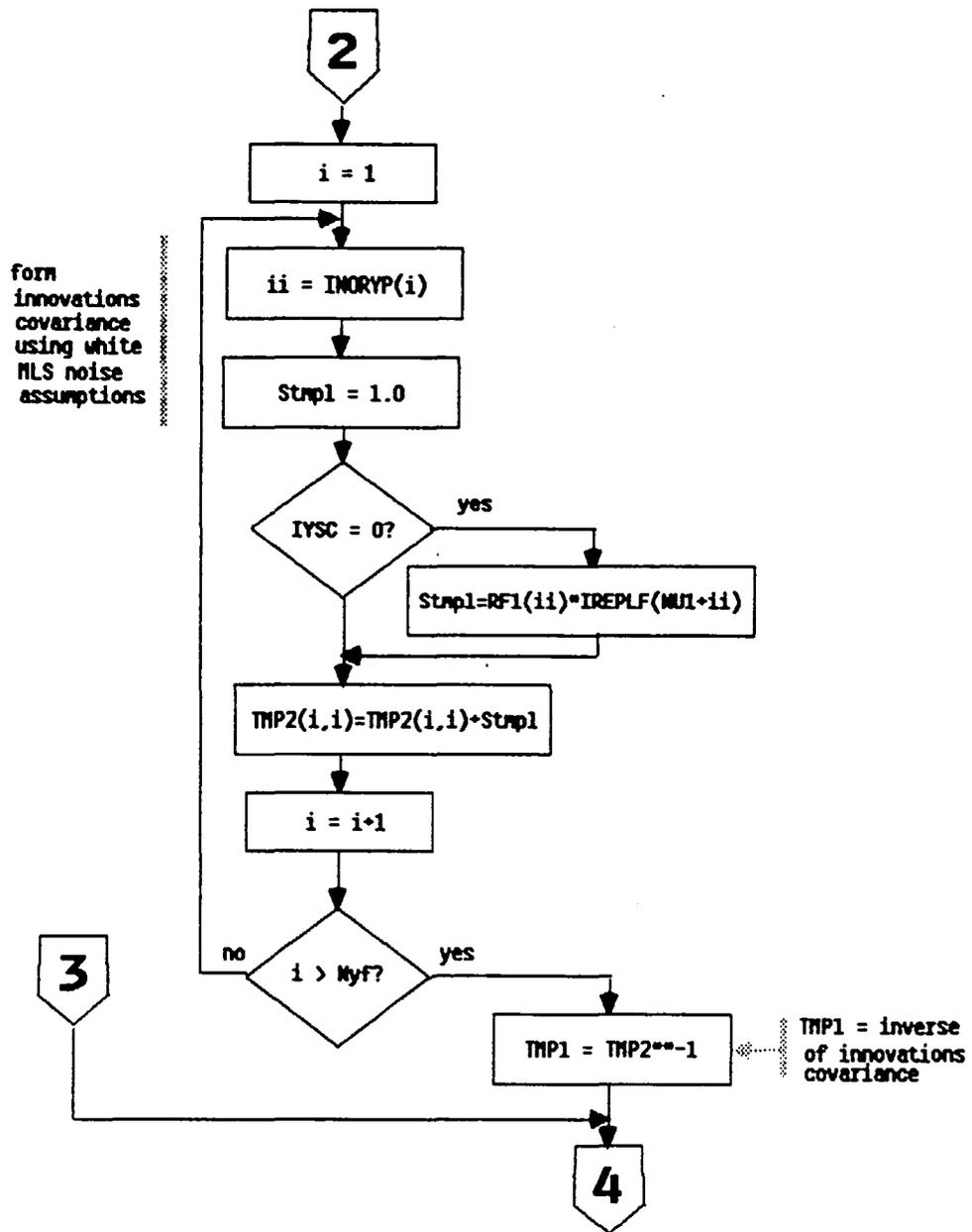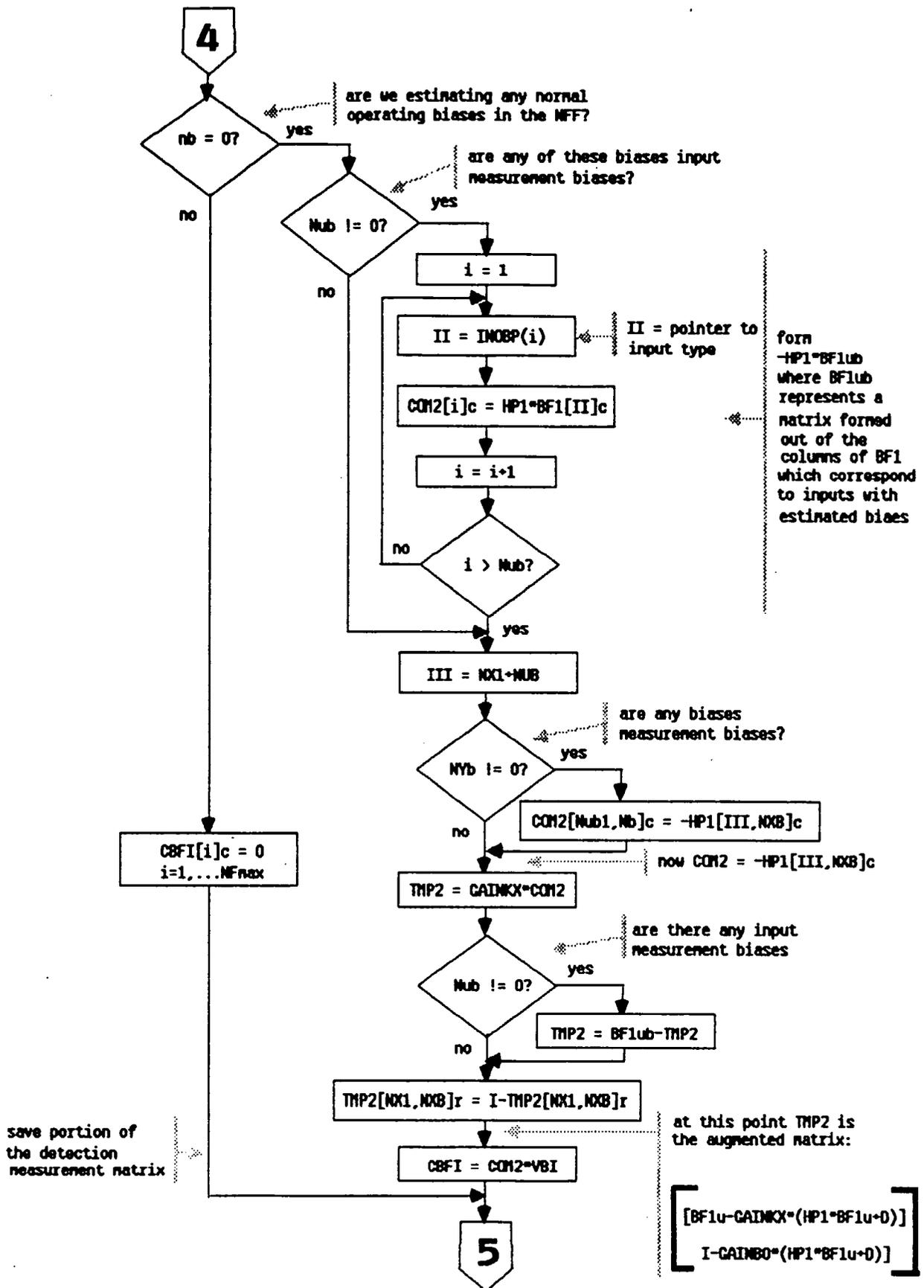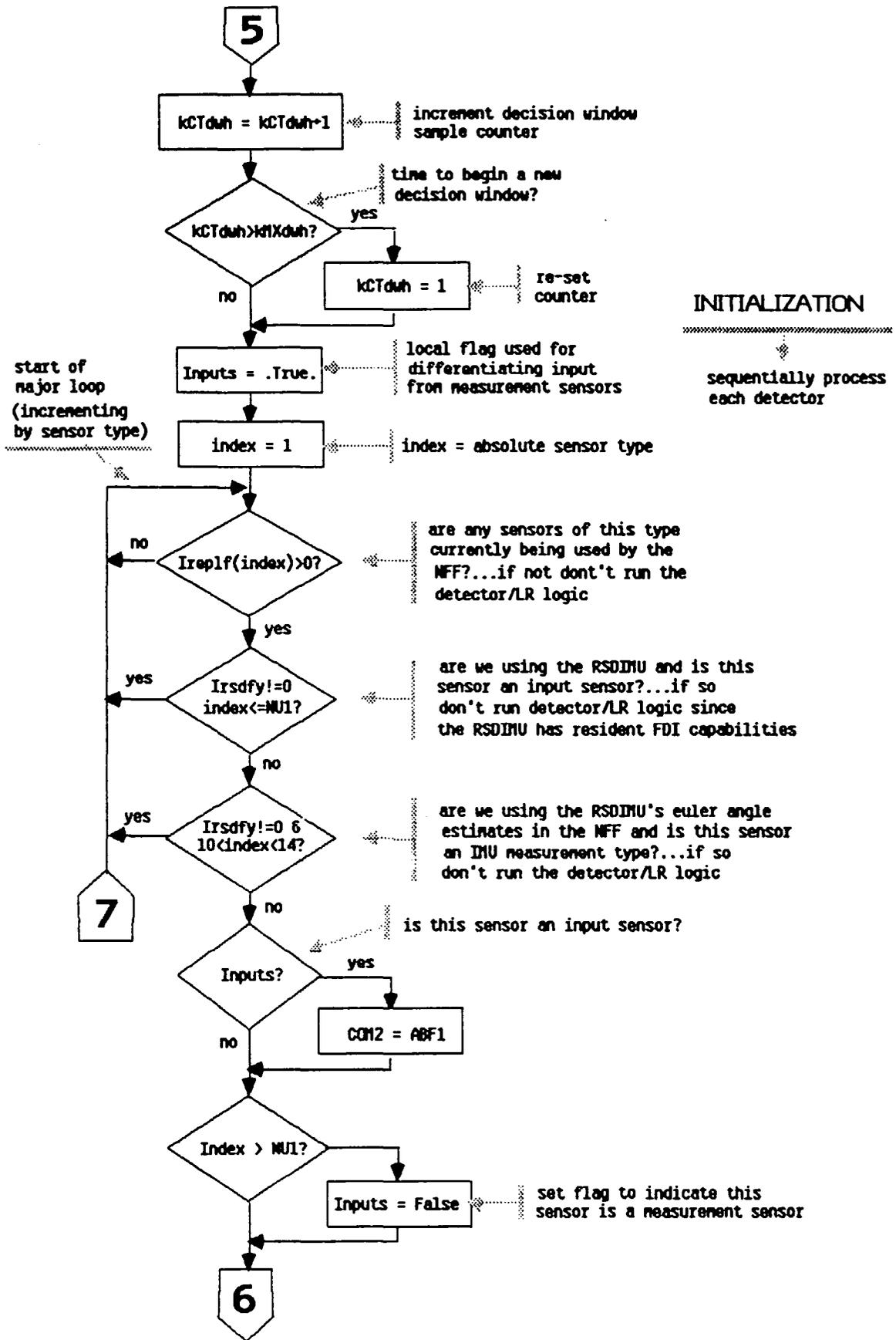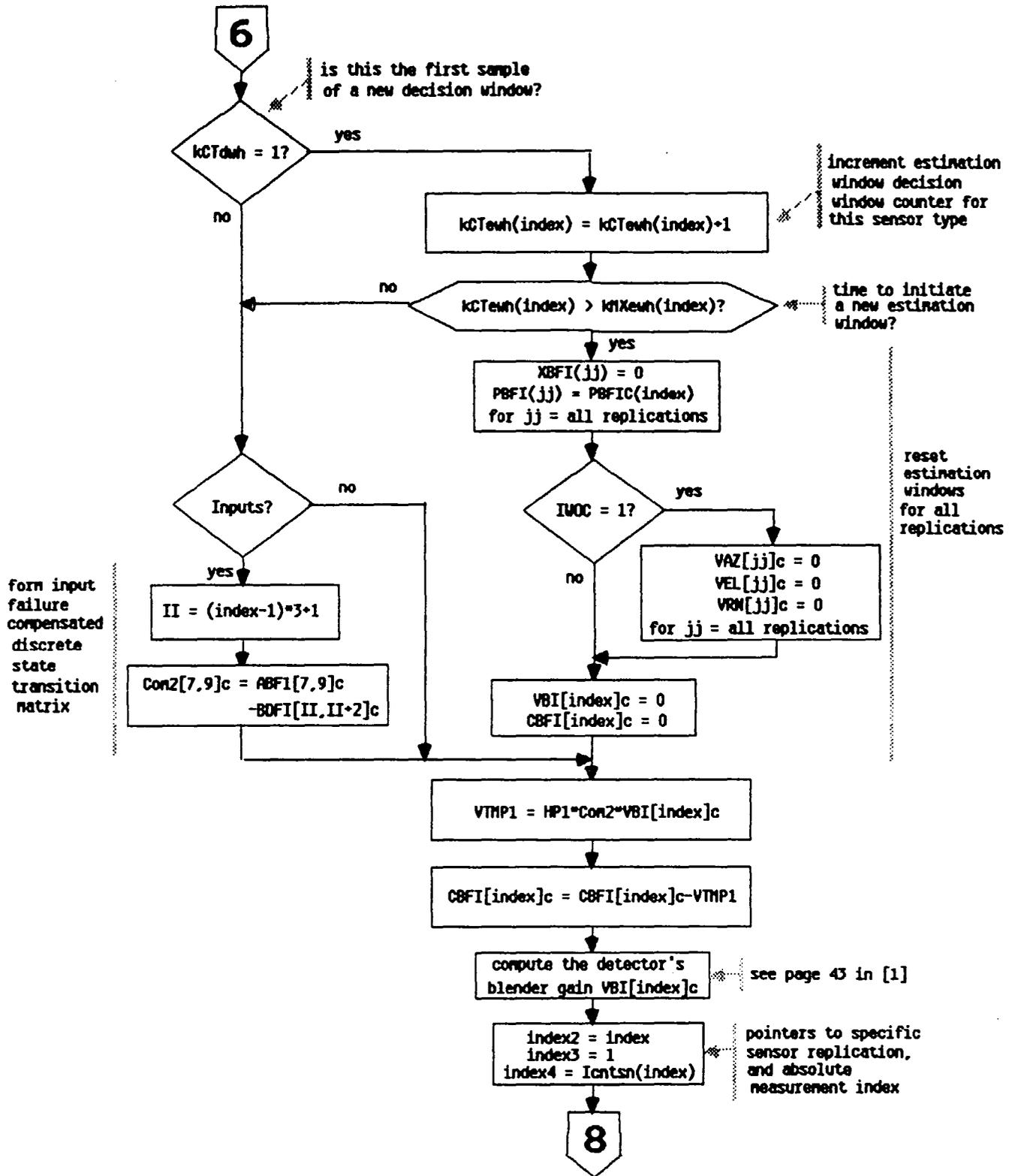**Figure 8. Flow Diagram for Subroutine DETECT (continued)**

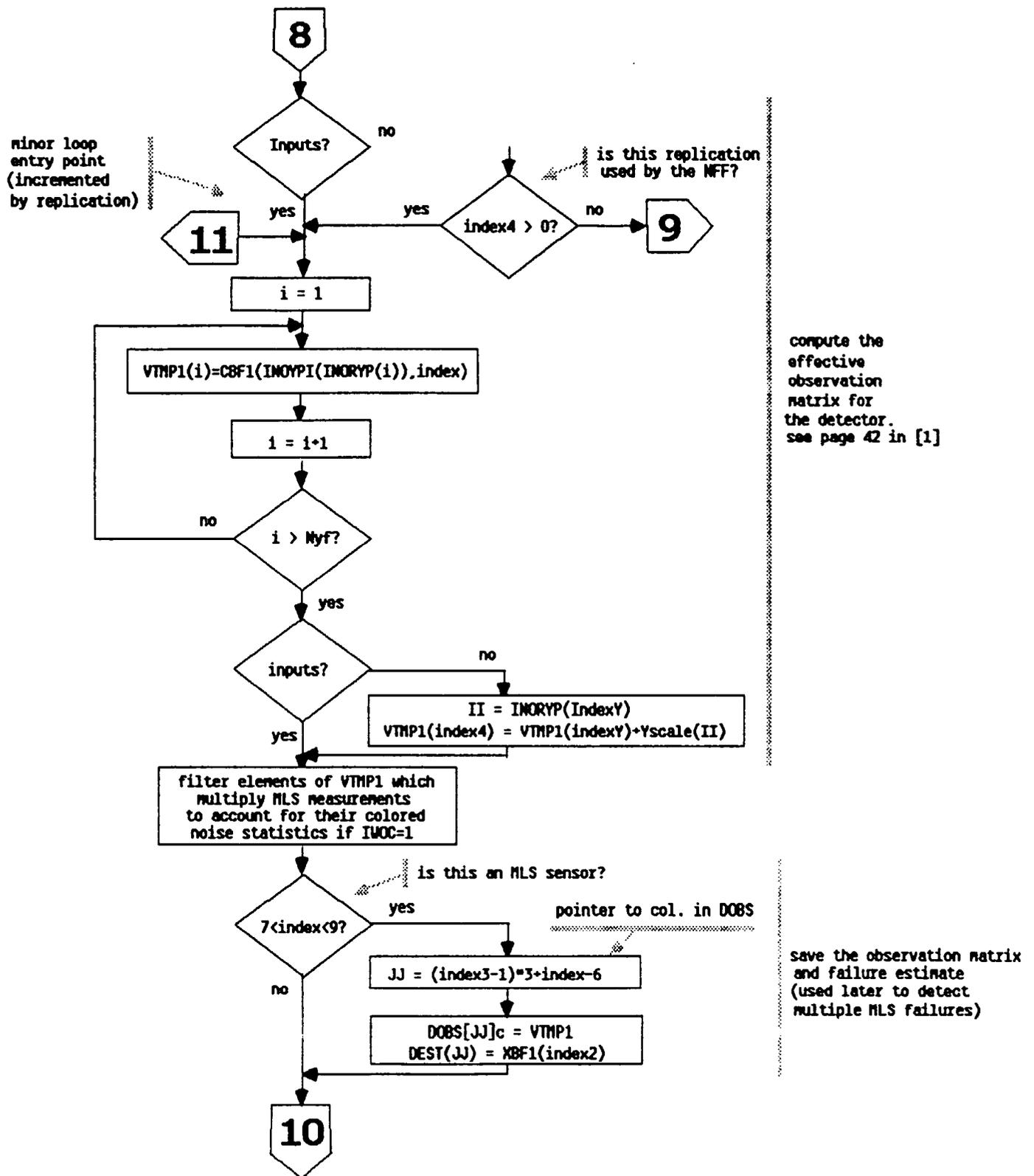Figure 8. Flow Diagram for Subroutine DETECT (continued)

58

Figure 8. Flow Diagram for Subroutine DETECT (continued)

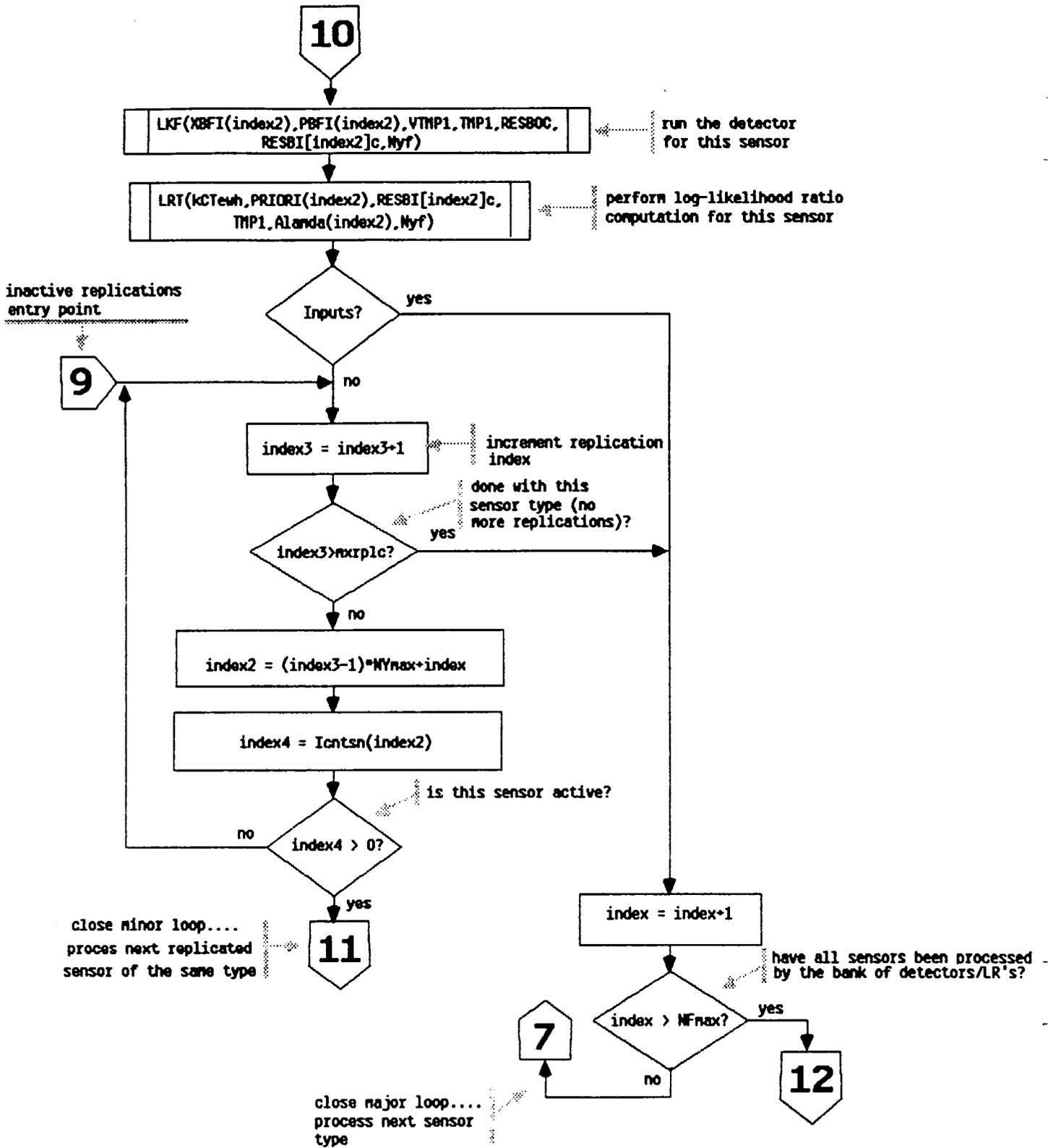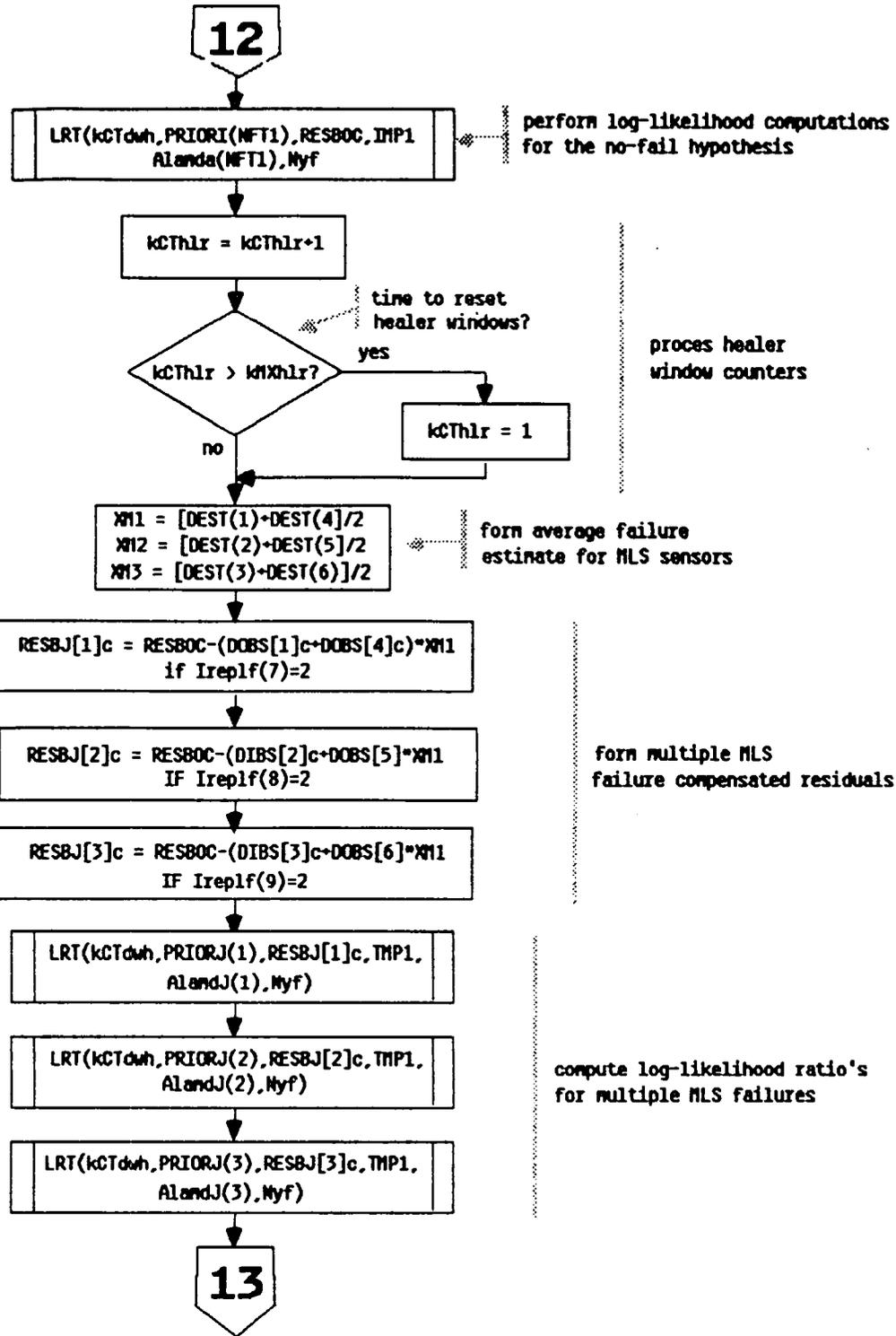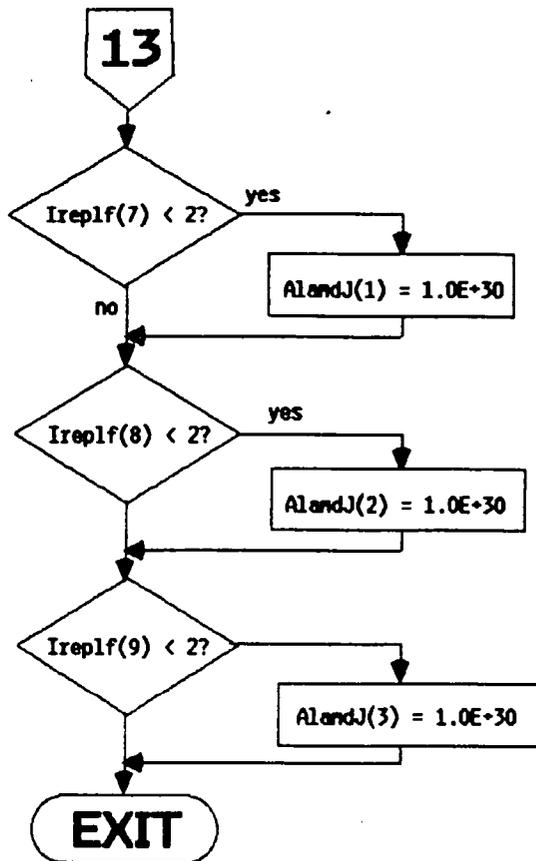Figure 8. Flow Diagram for Subroutine DETECT (continued)

60

```
       ┌────┐
       │ 12 │
       └────┘
          │
          ▼
┌──────────────────────────────┐      ⋮ perform log-likelihood computations
│ LRT(kCTdwh,PRIORI(NFT1),RES80C,IMP1  ⋮ for the no-fail hypothesis
│ Alanda(NFT1),Myf              │
└──────────────────────────────┘
          │
          ▼
     ┌──────────────┐
     │ kCThlr = kCThlr+1 │
     └──────────────┘
          │                  ⋮ time to reset
          ▼                  ⋮ healer windows?
        ╱──────────╲                              proces healer
       ╱            ╲         yes                 window counters
      ◄  kCThlr > kMXhlr? ────────────┐
       ╲            ╱                 │
        ╲──────────╱                 ▼
          │ no              ┌──────────────┐
          │                 │ kCThlr = 1   │
          ▼◄────────────────┴──────────────┘
┌──────────────────────────────┐
│ XM1 = [DEST(1)+DEST(4)]/2     │   ⋮ form average failure
│ XM2 = [DEST(2)+DEST(5)]/2     │   ⋮ estimate for MLS sensors
│ XM3 = [DEST(3)+DEST(6)]/2     │
└──────────────────────────────┘
          │
          ▼
┌──────────────────────────────┐
│ RESBJ[1]c = RES80C-(DOBS[1]c+DOBS[4]c)*XM1 │
│ if Ireplf(7)=2                │
└──────────────────────────────┘
          │
          ▼
┌──────────────────────────────┐
│ RESBJ[2]c = RES80C-(DIBS[2]c+DOBS[5]*XM1   │   form multiple MLS
│ IF Ireplf(8)=2                │               failure compensated residuals
└──────────────────────────────┘
          │
          ▼
┌──────────────────────────────┐
│ RESBJ[3]c = RES80C-(DIBS[3]c+DOBS[6]*XM1   │
│ IF Ireplf(9)=2                │
└──────────────────────────────┘
          │
          ▼
┌──────────────────────────────┐
│ LRT(kCTdwh,PRIORJ(1),RESBJ[1]c,TMP1,  │
│ AlandJ(1),Myf)                │
└──────────────────────────────┘
          │
          ▼
┌──────────────────────────────┐
│ LRT(kCTdwh,PRIORJ(2),RESBJ[2]c,TMP1,  │   compute log-likelihood ratio's
│ AlandJ(2),Myf)                │           for multiple MLS failures
└──────────────────────────────┘
          │
          ▼
┌──────────────────────────────┐
│ LRT(kCTdwh,PRIORJ(3),RESBJ[3]c,TMP1,  │
│ AlandJ(3),Myf)                │
└──────────────────────────────┘
          │
          ▼
       ┌────┐
       │ 13 │
       └────┘
```

Figure 8. Flow Diagram for Subroutine DETECT (continued)

Figure 8. Flow Diagram for Subroutine DETECT (concluded)

name: LKF

func: LKF provides the estimator structure for the failure
        detectors. LKF implements a linear Kalman filter using the
        information form, and assumes a scalar state equation. The
        module functions as follows:
        The plant equation is:
                Xmi(k+1) = Xmi(k) ;   i.e. no dynamics
        the measurement equation is of the form:
                Y(k+1) = Ci*Xmi(k) + n(k+1) ;   Y(k+1) is a vector
                of nv elements
                where n(k+1) ~ N(0,RT)
        the filter equations are:
                RTinv = RT**-1 (measurement noise covariance)
                Gi = [1.0/Pmi(k)]*Ci'*RTinv (filter gain)
                Pmi(k+1) = Pmi(k) + Ci'*RTinv*Ci (filter information)
        where, remember, Pmi is defined as the inverse of the
        estimation error covariance (i.e. the information matrix)
                Ri(k+1) = Y(k+1) - Ci*Xmi(k) (detector residuals)
                Xmi(k+1) = Xmi(k) + Gi*Ri(k+1) ("best" estimate)
        For a more detailed explanation of the detectors
        implementation see section 2.4 in [3], and section 2.1.2
        in [2].

call: Call LKF (Xmi,Pmi,Ci,RTinv,Y,Ri,nv)

args: Xmi -   double   inout   scalar estimate of the state (i'th
                         failure level estimate)
        Pmi -   double   inout   scalar filter information matrix
                         (information in i'th failure estimate Xmi)
        Ci -   double   in   effective observations matrix
                         (computed in DETECT)
        RTinv - double   in   inverse of the measurement noise
                         covariance matrix (NFF innovations covariance)
        Y -   double   in   observations vector (expanded
                         innovations from the no-fail filter)
        Ri -   double   out   innovations sequence from the LKF
                         (failure compensated innovations sequence)
        nv -   integer in   number of elements in the observations
                         vector, Y

refs: None
refby: DETECT
comm: MAIN1

63

name: LRT
func: LRT computes the log likelihood ratios over a decision window.
         The computations are as follows:
         1) if k=1 A=-PHj. This initializes the log likelihood ratio,
            A, to -ln(PHj) at the start of a new decision window.
            (Subroutine INITG initially stores PHj as the log of the
            a-priori probability of a sensors failure).
         2) SUMI = RES'*RTinv*RES
         3) A = 0.5*SUMI + A
         The reader should refer to section 2.7 in [3] or section 2.1.4
         in [2] for a more detailed description of this method.
call: Call LRT (k,PHj,RES,RTinv,A,ny)
args: k -   integer in   decision window simulation step
                  counter
         PHj -   double   in   logarithm of the a-priori probability
                  that the j'th sensor will fail
         RES -   double   in   failure corrected innovations sequence
                  from the j'th failure detector
         RTinv - double   in   inverse of the innovations covariance
                  matrix
         A -   double   inout   computed value of log likelihood ratio
                  for the j'th failure hypotheses
         ny -   integer in   number of observations
refs: MAT3A
refby: DETECT
comm: MAIN1

name: DECIDE
func: DECIDE computes the decision cost which minimizes Bayes Risk,
         and chooses the most likely hypothesis conditioned on this
         cost vector. DECIDE considers singleton sensor failures as
         well as dual simultaneous failures in MLS sensors. DECIDE
         operates as follows:
         1) find the smallest log likelihood ratio for singleton
            failures (stored in A)
         2) find the smallest log likelihood ratio for multiple
            failures (stored in ALAMDj)
         3) pick the smallest of 1) or 2) and determine the
            corresponding sensor type(s) and replication(s) of this
            sensor

4) if the chosen hypothesis is the no-fail condition
simply return:
5) otherwise, print out various messages informing the user
of the decision, and update the failure queue (i.e. NNfail,
IfailT, and IfailR)

call: Call DECIDE (m,cost,A,Beta)

args: m -   integer in   total number of singleton failures +1
(representing the no-fail hypothesis)

cost - double in   currently not used

A -   double in   vector of log likelihood ratios for
singelton failures

Beta - double in   currently not used

refs: CONVRF, MXMN2, TLOUT, VMPRT

refby: NAV

comm: DCIDEI, DETINF, DETXBI, FILTRT, HFCOM, INOU, MAIN2,
MULTDT, NAMES, PLOTS, SETCOM, SIMCOM, SYSU1

name: RECONF

func: RECONF reconfigures the FTS for proper operation (if possible)
after failures have been detected and isolated, and after
sensors heal. Figure 9 shows a functional flow diagram of
the operation of RECONF. Also, see section 2.6 of [2].

call: Call RECONF (IHfail,Iabort)

args: IHfail- integer in   Heal/Fail reconfiguration flag where
IHfail=1 for failures, and otherwise for healings

Iabort- integer out   run abort flag where Iabort=0
indicates normal operation, and otherwise indicates
the run should be aborted

refs: ALTYP, CLPSIO, EQUATE, IMTCG2, KALMAN, MATCG2, MATNUL, MINSET
MSCALE, MTH$DEXP, MTH$DLOG, NOISR, PNTINV, RCOV, RESCMP,
SETISN, SUMIN, TLOUT

refby: CHKRAD, NAV

comm: DCIDEI, DETINF, DETXBI, EKBFO, EKF1, FILTRT, GBLEND, HEALCM,
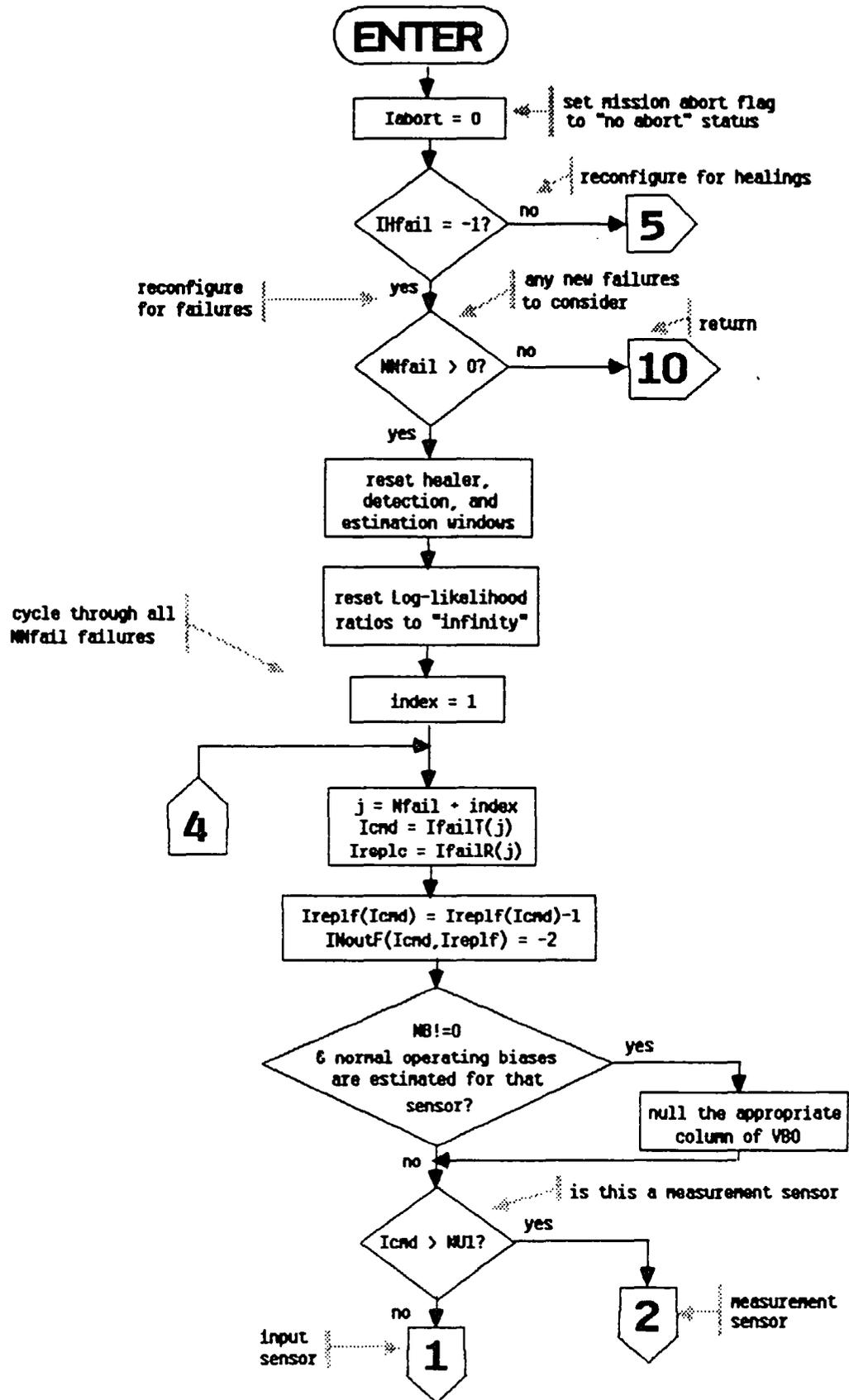HFCOM, INITVL, SETCOM, SIMCOM, SYSU1, SYSX1, SYSXBO, SYSYBO

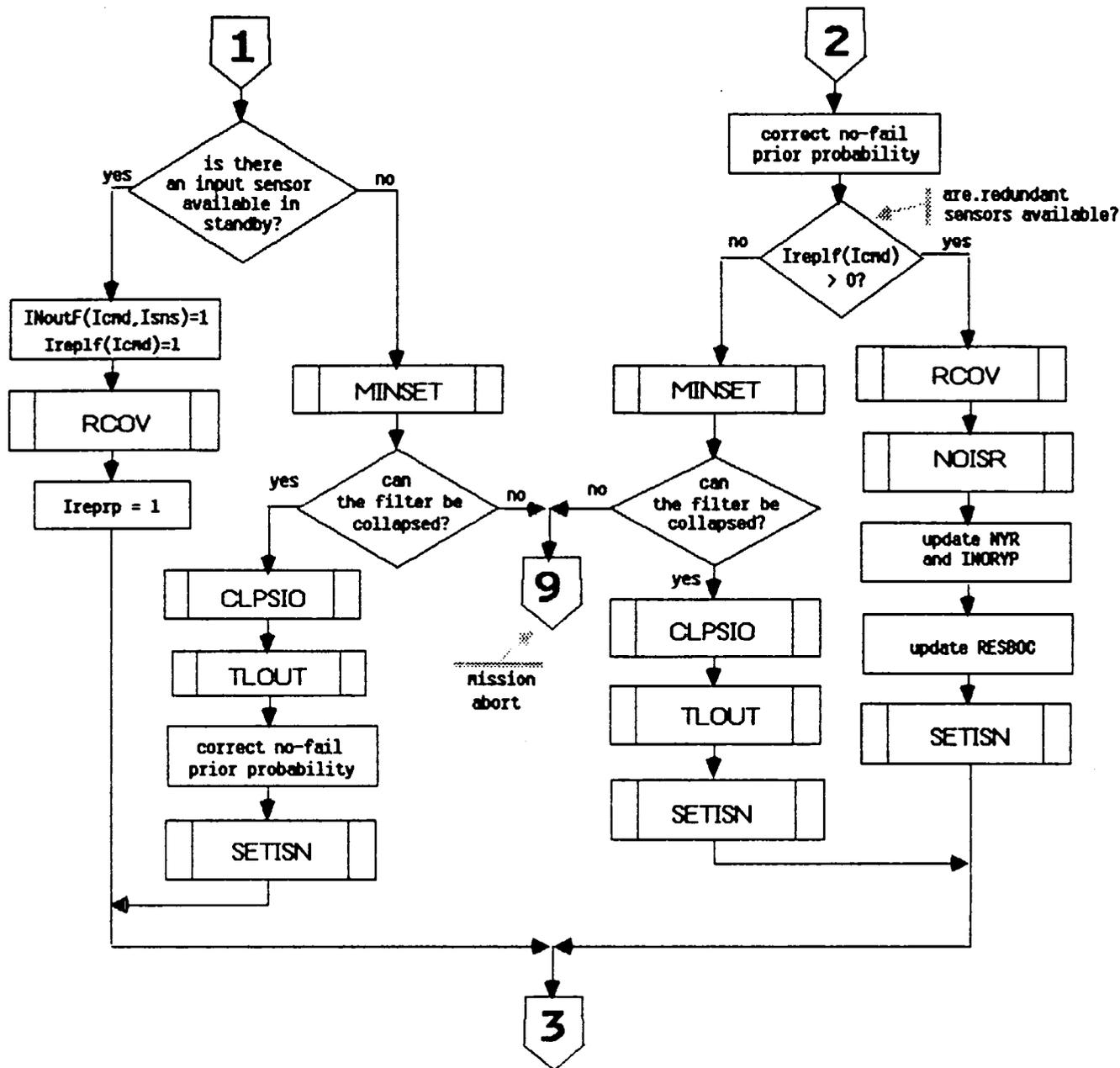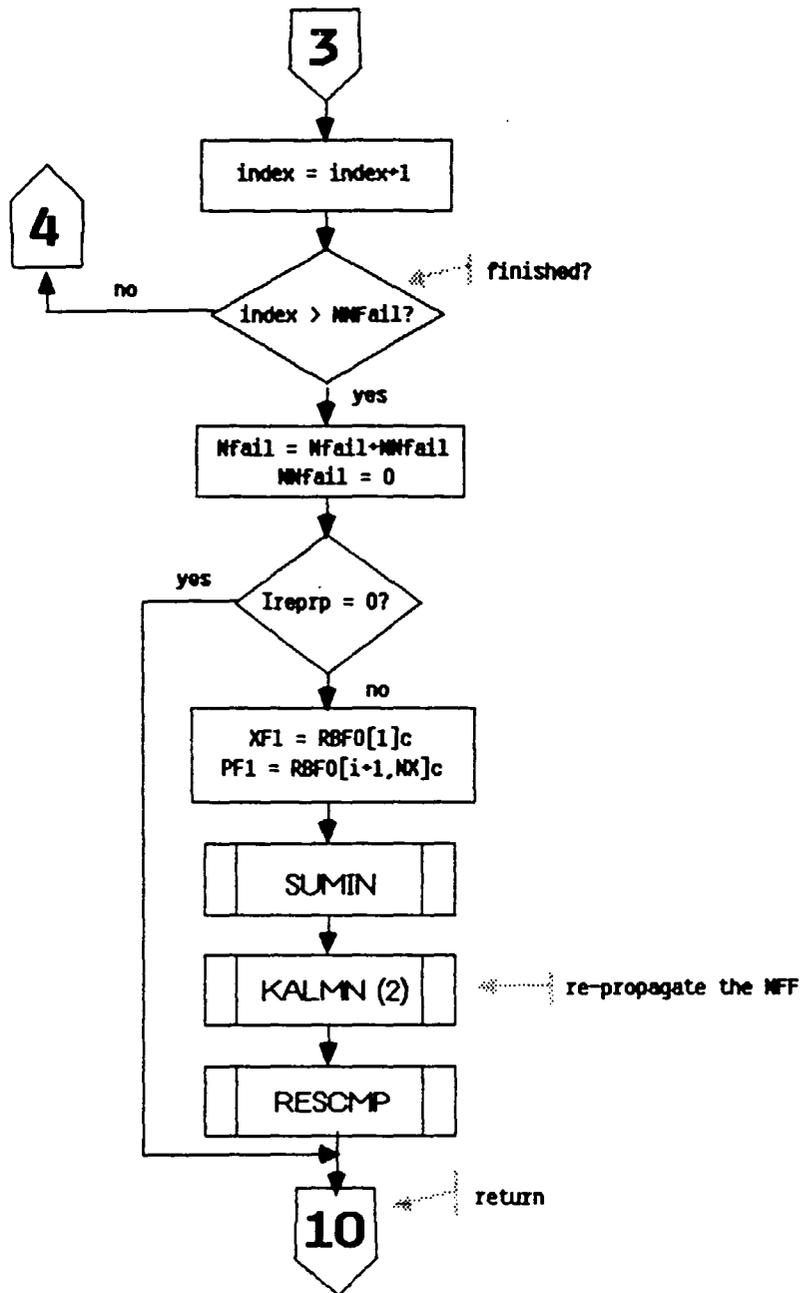Figure 9. Flow Diagram for Subroutine RECONF

66

Figure 9. Flow Diagram for Subroutine RECONF (continued)

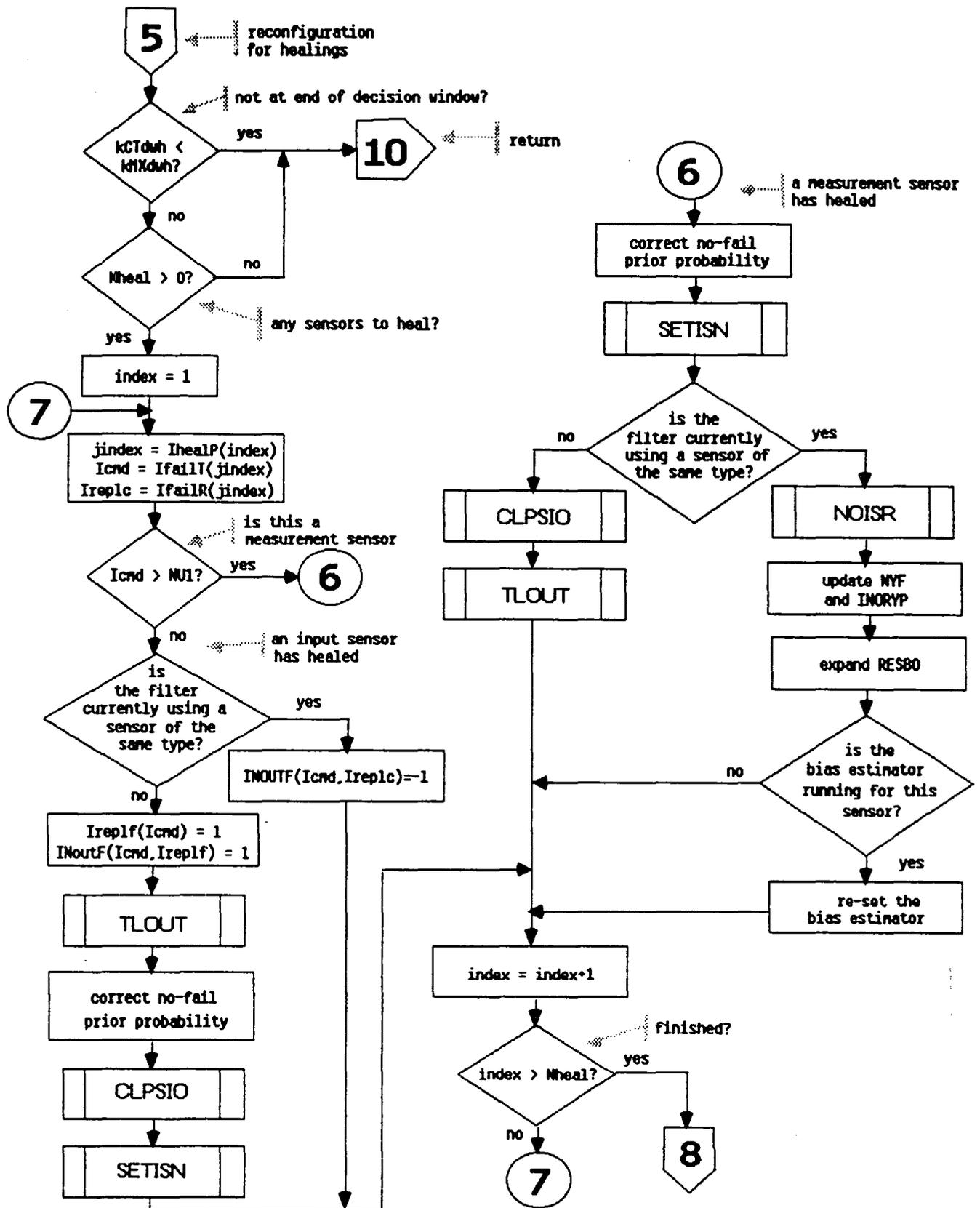Figure 9. Flow Diagram for Subroutine RECONF (continued)

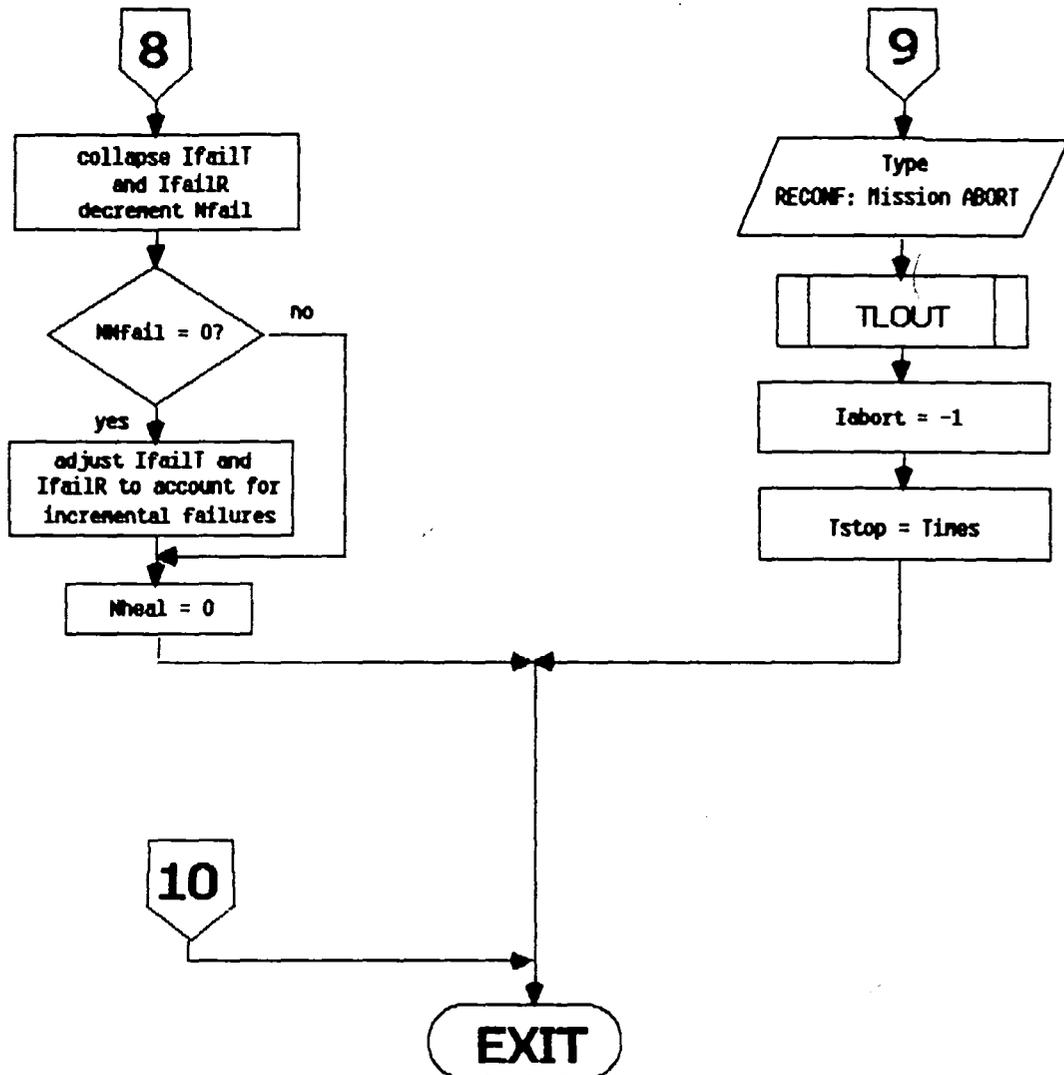Figure 9. Flow Diagram for Subroutine RECONF (continued)

Figure 9. Flow Diagram for Subroutine RECONF (concluded)

name: CLPSIO
func: CLPSIO is used to collapse (or expand) the no-fail filter and
         its associated data structures due to a single failure
         (healing) of a sensor. In particular, CLPSIO does the
         following:
         1) If Iclps<0 (i.e. collapse no-fail filter)
             a) if Isns<=NU1 (i.e. for input sensors)
                NOTE: currently FINDS doesn't allow input sensors to
                be removed. The logic that is used currently is only
                partially complete.
                    * set OF1(Isns)=0.0
                    * reset PF1 and PBF0 by calling subroutine RCOV
                    * collapse the input mapping vector, INOUP and adjust
                      NUIC
                    * if NB!=0 and this sensor has a normal operating bias
                      being estimated, collapse the bias estimator by
                      calling subroutine CLPSBE
             b) if Isns>NU1 (i.e. for measurement sensors)
                    * set RF1(ICmdY)=0.0
                    * reset PF1 and PBF0 by calling subroutine RCOV
                    * update NY and INOYP
                    * update NYF and INORYP
                    * collapse the residuals vector, RESBOC
                    * update the inverse measurement pointing vector,
                      INOYPI
                    * if NB!=0 and the no-fail filter is estimating a
                      normal operating bias for this sensor - collapse the
                      bias portion of the filter by calling subroutine
                      CLPSBE
         2) If Iclps>=0 (i.e. expand the no-fail filter
             a) for input sensors:
                    * reset the process and measurement noise matrices
                      OF1 and RF1, by calling subroutine NOISR
                    * update NUIC and INOUP
                    * if a normal operating bias is to be estimated add it
                      via subroutine CLPSBE
             b) for output sensors:
                    * call NOISR to set OF1 and RF1
                    * update NY, and INOYP
                    * update NYF, and INORYP
                    * expand RESBOC
                    * update INOYPI
                    * if NB!=0 and a normal operating bias is to be
                      estimated - call CLPSBE

71

```
                    * correct the partial derivative of h w.r.t. x, HP1
                      by calling subroutine UPDPH
call:    Call CLPSIO (Iclps,Isns,Ireplc)
args:    Iclps - integer in      flag used to control the collapse/
         expansion of the no-fail filter, where Iclps=-1 indicates to
         collapse and Iclps=1 indicates to expand it.
         Isns -  integer in       absolute index of the sensor
         Ireplc- integer in       replication of the sensor
ints:    IcmdY - integer          absolute measurement sensor index as
                 described in Table 1 on page 9 in [1]
         IRsns - integer          absolute replicated sensor index (see
                 Table 6).
refs:    ALTYP, CLPSBE, IMTCG2, MATCG2, NOISR, PNTINV, RCOV, UPDPH
refby:   RECONF
comm:    DETINF, DETXBI, EKBFO, EKF1, FILTRT, INITVL, SYSU1, SYSX1,
         SYSXBO, SYSYW1
```

```
name:    NOISR
func:    NOISR resets the process or measurement noise covariance terms
         in the no-fail filter for a given sensor type.  In particular:
                    * if Isns <= NU1 (i.e. it corresponds to an input
                      measurement to the no-fail filter) and if body
                      mounted sensors are used:
                          QF1(Isns) = sig(Isns)**2
                      otherwise if the RSDIMU is used:
                          QF1(Isns) = sig(18)**2  for 1<Isns<3
                      or  QF1(Isns) = sig(17)**2  for 4<Isns<6
                    * if Isns > NU1 (i.e. a measurement sensor)
                          RF1(Isns) = sig(Isns-NU1)**2/Ireplf(Isns)
                      (remember Ireplf(Isns) is the number of active
                      sensors of this type currently used by the no-fail
                      filter)
call:    Call NOISR (Isns,Ireplc)
args:    Isns -  integer in       absolute index of sensor
         Ireplc- integer in       currently not used
refs:    None
refby:   CLPSIO, RECONF
comm:    ASOUT, FILTRT, SIGTAU, SYSU1, SYSYW1
```

name: RESCMP
func: RESCMP computes the expanded residuals sequence from the
         collapsed residual sequence generated by the no-fail filter.
         The results are then stored in RESBOC. This sequence is
         the same as the one which would have been generated had the
         filter been driven by all replications of the measurement
         sensors rather than their average value. See section 2.3.1
         in [2]. RESCMP also computes an estimate of the filter
         observations and stores this result in HXKP1.
call: Call RESCMP
args: None
refs: None
refby: NAV, RECONF
comm: AGOUT, ASOUT, CNTROL, DETINF, EKBFO, EKF1, FILTRT,
         MLOUT, PSIR, RAOUT, RIOUT, SYSU1, SYSXBO, YOBSRV

name: FILCOL
func: FICOL estimates colored MLS noise states and compensates the
         expanded innovations sequence generated by RESCMP. This is
         done in an effort to "whiten" the innovations sequence, since
         it is known that the MLS sensors have colored rather than
         white noise statistics.
call: Call FILCOL
args: None
refs: EQUATE, MTH$DEXP
refby: NAV
comm: CNEST, COLFIL, DETINF, FILTRT, FLTCTL, MAIN1, MLOUT, PJUNCK,
         SENSCM, SYNC, SYSU1

name: CLPSBE
func: CLPSBE is responsible for resetting the bias estimator portion
         of the no-fail filter such that a single bias can be added or
         deleted. In particular, CLPSBE:
         1) calls ADJSTBP to determine IBkey and IYkey and to adjust
            the bias pointer vector INOBP, as well as NXB, NUB, NYB,
            NUB1, and NB.

73

2) if IBkey<0 (implying that either the bias exists and we've
   tried to add it, or the bias doesn't exist and we've tried
   to delete it) then CLPSBE fails by printing out this
   message on the terminal:
       CLPSBE: Routine Fails
3) if kflag=-1 (collapse the bias estimator)
   a) the IBkey row and column of the bias filter error
      covariance, PBFO, is deleted
   b) the IBkey column of the bias filter blender gain, VBO,
      is deleted
   c) the IBkey row of the vector of bias estimates, XBFO,
      is deleted
4) if kflag!=-1 (expand the bias estimator)
   a) PBFO is expanded about the IBkey row and column, and
      they are zeroed out
   b) The initial value of the bias fiter error covariance is
      loaded into the appropriate diagonal element s.t.
          PBFO(IBkey,IBkey)=PBFOI(Ibias)**2
   c) VBO is expanded about the IBkey column, and it is zeroed
      out
   d) XBFO is expanded about the IBkey row and zeroed out

call:	Call CLPSBE (kflag,Ibias)
args:	kflag - integer in flag to indicate whether to collapse or expand the bias filter. (-1 => delete, +1 => add)
	Ibias - integer in absolute index of bias type to be added or deleted
ints:	IBkey - integer pointer to the location of the bias (absolute sensor index) "Ibias" in the reduced no-fail filter bias vectors and matrices
	IYkey - integer pointer to the location in the no-fail filter measurement vector which corresponds to bias "Ibias"
refs:	ADJTBP, ALTYP, MATCG2
refby:	CLPSIO
comm:	EKBFO, GBLEND, INITVL, SYSU1, SYSX1, SYSXBO, SYSYW1, YOBSRV

name: ADJTBP
func: ADJTBP increments or decrements various and
         scalars used by CLPSBE and the bias filter itself, when adding
         or deleting biases in the estimator.
call: Call ADJTBP (Iflag,Index,Irkev,Iykey)
args: Iflag - integer in   flag indicating addition or deletion
                 of a bias, where -1 => delete. and +1 => add
         Index - integer in   absolute index to sensor type of
                 bias to be added or deleted
         Irkey - integer out   to bias in the reduced
                 bias set (i.e. the bias vectors used by the filter)
                 if the routine succeeds and Irkey=-1 if the routine
                 fails
         Iykey - integer out   to absolute output
                 index which corresponds to the bias referred to by
                 Index. If the bias is on an input measurement then
                 Iykey=0
refs: ALTYP, IMTCG2, PNTINV
refby: CLPSBE
comm: CMPSTF, DETXBI, SYSU1, SYSX1, SYSXB0, SYSYW1

name: RCOV
func: RCOV resets the no-fail filter's estimation error covariances
         once a failure has been detected and isolated. In particular
         it sets:

$$1) \quad \begin{pmatrix} P0 & 0 \\ 0 & Pb \end{pmatrix} = \begin{pmatrix} P0 & 0 \\ 0 & Pb \end{pmatrix} + Vi*Vi'*X**2$$

         where $X**2 = Xmi**2 + 1.0/Pmi$
         2) if PBF0 > Pbf0I then PBF0 is set to Pbf0I (i.e. the initial
            uncertainty) and XBF0 = 0.0.
         This corresponds to the conditional covariance reset method
         discussed in section 2.6 of [2]. Logic still exists in this
         routine which can be used to cause the other two reset methods
         discussed in [2] to be used. however, a re-compilation of the
         code would be required.
call: Call RCOV (P0,Pb,X0,Vi,Xmi,Pmi,Icmd)
args: P0 -   double inout   bias free filter estimation error

			covariance	
Pb	-	double inout	bias filter estimation error covariance	
XO	-	double inout	vector of current state and bias estimates	
Vi	-	double in	blender gain for the i'th detector	
Xmi	-	double in	estimate of the i'th failure level	
Pmi	-	double in	information matrix for the i'th failure	
Icmd	-	integer in	absolute sensor type of the failed sensor	

refs: MATNUL, UPDH, VMPRT
refby: CLPSIO, RECONF
comm: CMPSTF, EKBFO, EKF1, INITVL, INOU, MAIN1, SYSU1, SYSX1, SYSXBO, SYSYW1, YOBSRV

name: MINSET
func: MINSET determines if the A/C can operate if a particular
         sensor is removed. by knowing the minimal sensor sub-sets
         allowed for stability. Currently MINSET will allow all
         replications of IMUs to be removed. all replications of MLS
         elevation - provided the radar altimeter is available. and
         all replications of the radar altimeter - provided MLS
         elevation is available. Otherwise. if all replications of any
         other sensor are lost MINSET will abort the run.
call: Call MINSET (Isns,Ireplc,Iok)
args: Isns - integer in   absolute sensor index
         Ireplc- integer in   replication of the sensor - currently
                 not used
         Iok - integer out   abort/run flag where if
                Iok = -1 perform a misson abort, otherwise if
                Iok = 1 allow the sensor in question to be removed
                         and the run to proceed
refs: None
refby: RECONF
comm: FILTRT, LOGIC4

name: HEALR

func: HEALR manages the operation of the healer detection logic.
          Its primary function is to maintain all sensors selected
          as "failed" by the FDI logic and determine if they have
          healed or recovered. Healer decisions are made ONLY at the
          end of a healer decision window. Special logic is employed in
          order to force the IMUs to heal in a coordinated fashion
          (i.e. for the i'th replication of an IMU to heal; phi, theta,
          and psi must all be healthy). This logic, while specific to
          the IMUs, forms the framework required to impose arbitrary
          additional constraints on a sensor's healing. A detailed
          explanation of how the healers operate can be found in section
          2.5 in [2]. Also helpful in understanding this routine is the
          description of common blocks HEALCM and HFCOM. Figure 10
          provides a detailed flowchart to indicate how the healers are
          realized.

call: Call HEALR

args: None

ints: Xsum -   double   vector of running sums - one for each
                    active healing process. See subroutine AVECMP for a
                    description of how the elements are computed

          IfailP-   integer   where the row index is
                    the healer process number, and the value of each
                    element the corresponding failed sensor's
                    location in the list of failed sensors
                    (IfailT and IfailR)

          NfailL-   integer   length of IfailP, i.e. number of
                    healer processes currently running. Both IfailP and
                    NfailL are updated at the start of each new healer
                    window.

          IMUrep-   integer   local test vector used for IMU healing
                    logic. The row index is the IMU replication number,

and
                    the value is the number of sensors within that IMU

that
                    have healed

          IMUmap-   integer   local test matrix used for IMU healing
                    logic. The column index is the IMU replication

number.
                    the rows store particular sensors which have healed.
                    and the value stored is the corresponding position in
                    the list of healed sensors, IhealP. Note this matrix
                    allows us to map the locations in IhealP to each
                    replication of an IMU

Iremov- integer   local test vector used for IMU logic.
          This array stores the locations in IhealP which
          correspond to sensors which passed the healer decision
          criterion but cannot be allowed to heal - due to other
          constraints. Currently, this means only a portion of
          an IMU has healed
Nremov- integer   length of Iremov, i.e. number of
          sensors which must be removed from the list of healed
          sensors, IhealP
refs: AVECMP, BUBBLE, LRTHLR, TLOUT
refby: NAV
comm: HEALCM, HFCOM, SYSU1
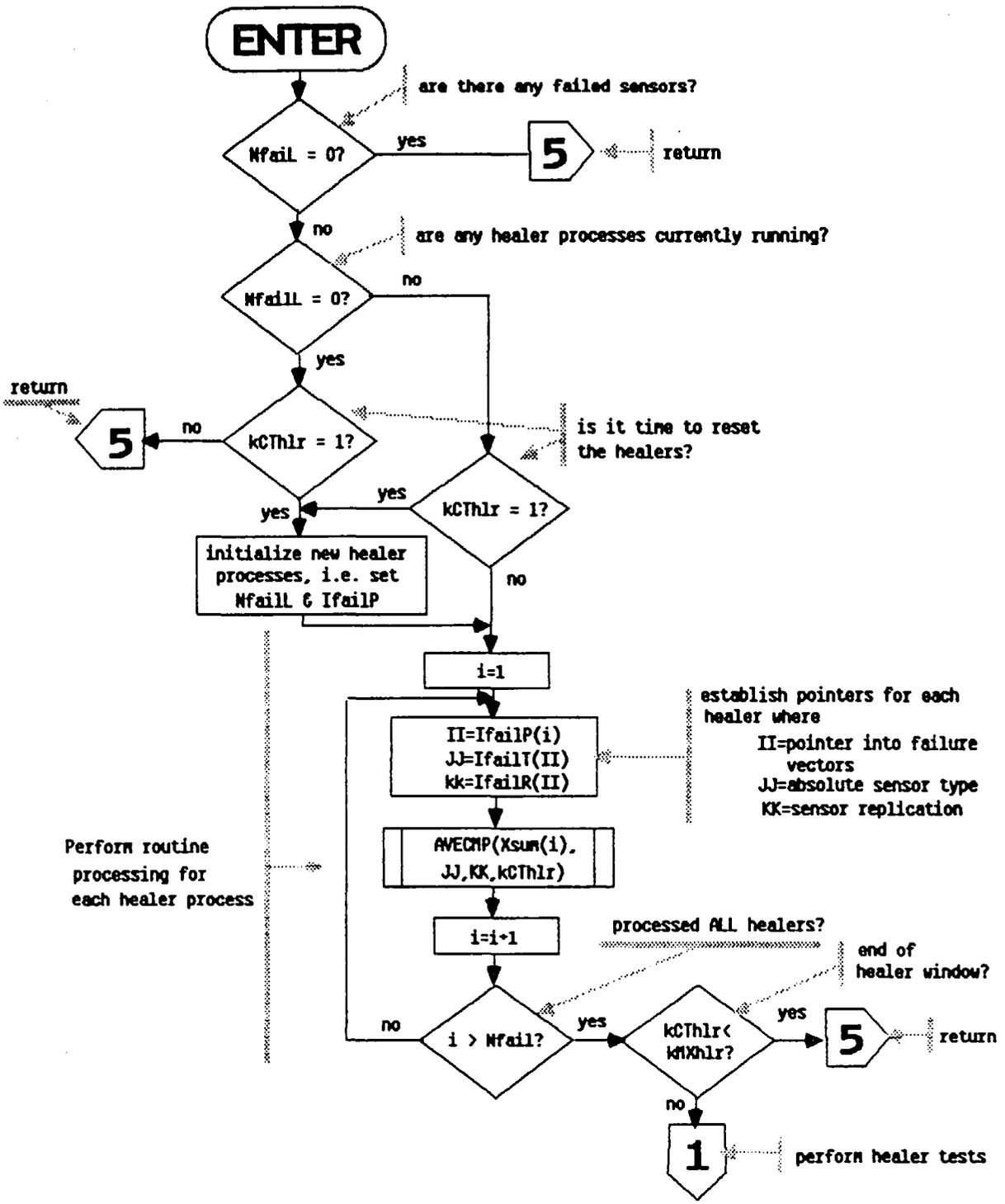
Figure 10. Flow Diagram for Subroutine HEALR
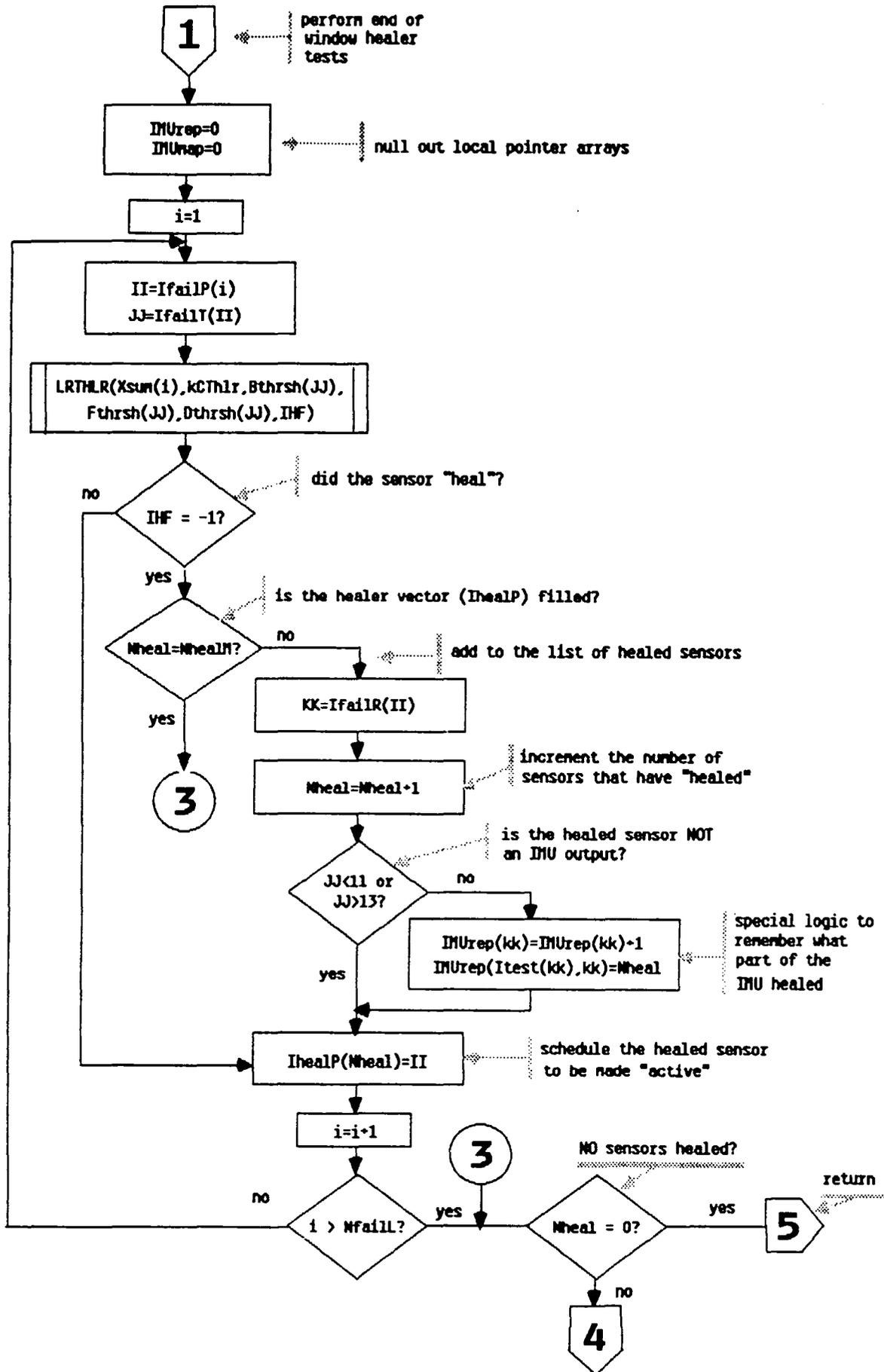
Figure 10. Flow Diagram for Subroutine HEALR (continued)
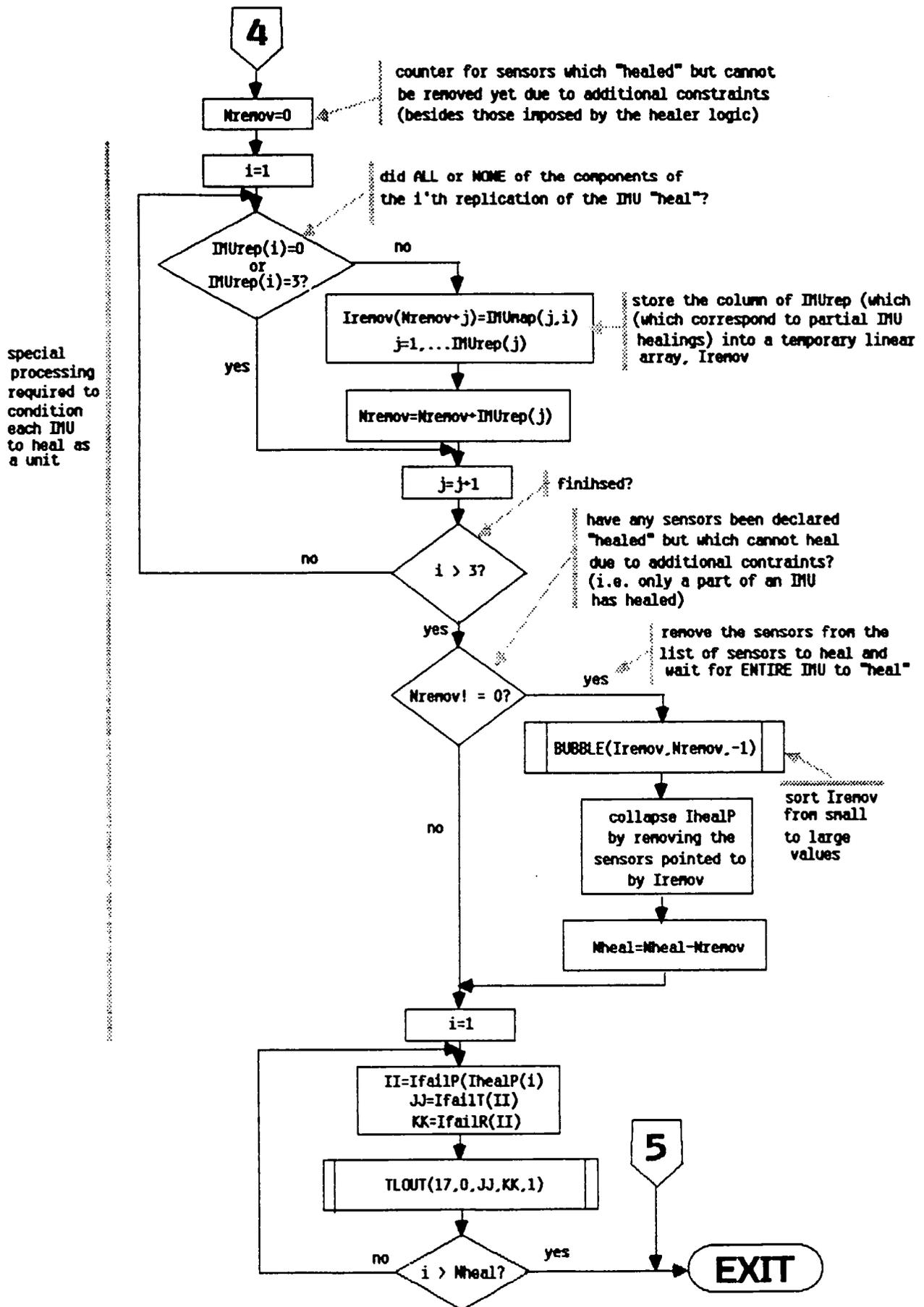
Figure 10. Flow Diagram for Subroutine HEALR (concluded)

81

name: AVECMP
func: AVECMP supports the operation of HEALR by computing the
        running sum of (Xwork-Xfail) over the healer window of
        length kmxhlr. The value of the sum is reset to zero at the
        start of a new healer window. Xwork and Xfail are defined
        as follows:
                    *   for input measurement sensors:
                        Xwork = a measurement from a (assumed) correctly
                                working replicated sensor of the same type
                                as the failed one
                        Xfail = a measurement from the failed sensor
                    *   for output measurement sensors:
                        Xwork = the estimate of the observation obtained
                                from the no-fail filter
                        Xfail = the measurement from the failed sensor.
call: Call AVECMP (Xsum,IfailT,IfailR,kreset)
args: Xsum - double inout   the running sum of (Xwork-Xfail) for
                a particular failed sensor
        IfailT- integer in   absolute sensor type of failed sensor
        IfailR- integer in   replication of failed sensor
        kreset- integer in   reset flag which indicates the start
                of a new healer window if kreset=1
refs: None
refby: HEALR
comm: AGOUT, ASOUT, EKF1, LAOUT, MLOUT, PSIR, RAOUT, RIOUT,
        RGOUT, SYSU1, YOBSRV

name: LRTHLR
func: LRTHLR performs a likelihood ratio test to determine if
        a sensor has healed at the end of a healer window. The
        test is performed as follows:
        1) a maximum likelihood estimate of the normal operational
            bias is computed as:
                Best = Xsum/length
            where Xsum is computed in AVECMP, and length is the number
            of samples in the window. The estimate is limited by:
                if Best > Bthrsh   then   Best = Bthrsh
                if Best < -Bthrsh   then   Best = -Bthrsh
            where Bthrsh is the largest expected bias level for this
            sensor type (set in INITG)

2) a maximum likelihood estimate for a failure level is computed as:

    Fest = Xsum/length

    The failure estimate is then limited by:

        if Fest > 0.0 & Fest < Fthrsh   then   Fest = Fthrsh
        if Fest < 0.0 & Fest > -Fthrsh   then   Fest = -Fthrsh

    where Fthrsh is the smallest expected failure level for this sensor type (set in INITG)

3) a decision function is evaluated as:

    xtmp = 2.0*(Fest-Best)*Xsum + length*(Best**2+Fest**2)

4) the value of the decision function is compared to a decision threshold, Dthrsh (set in INITG), and if xtmp is less than Dthrsh the sensor is declared "healed" (by setting IHF=-1). A detailed description of this method can be found in section 2.5 in [2].

Call:	Call LRTHLR (Xsum,length,Bthrsh,Fthrsh,Dthrsh,IHF)
args:	Xsum – double in the sum, over the entire healer window, of (Xwork-Xfail) as computed by AVECMP
	length– integer in the number of samples included in the healer window
	Bthrsh– double in the maximum expected value for a normal operating bias level on this sensor
	Fthrsh– double in the minimum expected value for a failure in this sensor
	Dthrsh– double in the decision threshold to be used in determining whether a sensor has healed
	IHF – integer out a flag indicating the outcome of the LRT. IHF=-1 if the sensor has healed and if IHF=0 it has not healed
refs:	DABS, DFLOAT
refby:	HEALR
comm:	None

name:	CONVRF
func:	CONVRF determines the proper conversion factor needed to convert from "program" engineering units to "user" or output units. It also supplies a 5 character literal name describing the name of the units. Currently only no-fail filter states and sensors are accounted for. The routine operates as follows:

83

1) the user supplies a flag, Iopt, to indicate whether states or sensors are to be considered
2) the user also supplies an index, n, which indicates in an absolute ordering convention found in table 1, page 9 in [1], which element of the state or sensor vector is desired.
3) CONVRF then determines the conversion factor required and stores it in the value of the function CONVRF, and the name of the units, stored in Lname.

call: X=CONVRF (n,Iopt,Lname)
args: n   -   integer in   absolute index into the state or sensor vector
       Iopt - integer in   a flag indicating n is an index into the no-fail filter state vector if Iopt=1; or n is a sensor type index if Iopt !=1
       Lname - char   out   a 5 character name for the units to be converted to.
       CONVRF- double out   the value of the conversion factor required to convert to user units.
refs: None
refby: DECIDE, INITG, PRNTIC, TLOUT
comm: MCONCO

name: AVBIAS
func: AVBIAS computes, for a particular sensor type, the effective average bias seen by the no-fail filter. AVBIAS operates as follows:
       1) for input sensors it subtracts the true signal and noise from each measurement and then, if the no-fail filter uses more than one replication, averages these quantities across replications. The true signal and noise are saved in the appropriate sensor modules in EFBSLA and EFBSRG for linear accelerometers and rate gyro's respectively.
       2) for output measurement sensors AVBIAS simply averages the true (i.e. simulated) bias levels across replications.
call: X=AVBIAS (n)
args: n   -   integer in   the absolute sensor type index found in table 1 on page 9 in [1]
       AVBIAS- double out   the effective average bias for the n'th sensor type

84

refs: SUMMER
refby: PRINTIC. SAVIT
comm: AGMP. ASMP. EFBS. FILTRT, LAMP. MLSMP. RALMP, RGMP

3.3.3 Documentation For File: FGAC.FOR –

3.3.4 Documentation For File: FWIND.FOR –

3.3.5 Documentation For File: FSENS.FOR -

3.3.6 Documentation For File: FIO.FOR -

name: SAVIT
func: To save a user selectable set of program variables in a
        periodic fashion on a binary plot file. SAVIT uses a 3 pass
        structure to provide this capability, where:
                Ipass=1 provides initialization - SAVIT interactively
                        prompts for groups of outputs to be saved
                Ipass=2 save (record) variables
                Ipass=3 flush buffers and close files
        The reader is directed to section 4.4 in [1] for a detailed
        discussion of the plot file contents and interactive prompts
        provided by SAVIT.
call: Call SAVIT
args: None
ints: Ipass - integer in   pass flag stored in common block
                FLTCTL
        Lsave - integer   vector of yes/no responses to the
                prompting questions (found in Table 6 in [7]) -
                used to control execution of the routine.
refs: ALTYP, AVBIAS, FILER1, ISPEC, LASK, MTH$DSQRT, RECRDS,
        SAVIT, SEQNCE, VECHG1, VECNUL
refby: FINDS
comm: ACCLS, AGMP, AGOUT, ANGLES, ARSTAT, ASMP, ASOUT, CMPSTF,
        CNEST, CNTROL, CONTRL, CRTE, DCIDEI, DETINF, DETXBI, DETYBI,
        EGUIDE, EKBFO, EKF1, FCOM1, FCOM2, FILTRT, FLTCTL, GSLOPE,
        GUIDE, GYROS, IMLS, INITVL, INOU, LAMP, LAOUT, LINAC, LOGIC4,
        MAIN1, MAIN2, MCONCO, MLOUT, MLSALL, MLSMP, PJUNK, PORT,
        PSIR, RALMP, RGMP, RGOUT, RGUIDE, RGYRO, RIOUT, RIOUT2,
        SPCFOR, SYSU1, SYSX1, SYSXBO, SYSYW1, VARLAT, VARLON, UPDAT,
        WIND, YOBSRV

name: PRNTIC

func: To print FINDS Filter-detector-healer initial conditions, as
          well as sensor module simulated normal operating parameters
          and scheduled failure information. The output is printed to
          a user specified ASCII file in a special table format. The
          output is printed in three passes - corresponding to different
          types of information - controlled by an input flag, IoptnZ.
          The reader is referred to figure 8 starting on page 66 of [1]
          for an example of the output generated by PRNTIC. See also
          discussion on IoptnZ below.

call: Call PRNTIC (kdsk,IoptnZ)

args: kdsk -   integer in   fortran unit number of the ASCII
                   file output will be directed to

          IoptnZ- integer in   output control flag where if:
                   * IoptnZ = 1:   print page 66 of [1] - except for the
                                    last two lines
                   * IoptnZ = 2:   print last two lines on page 66
                                    in [1] and tables 1, and 1a
                   * IoptnZ = 3:   print tables 2-5 on pages 68-69
                                    in [1]

refs: AHEDR, AVBIAS, CONVRF, FOR$DATE_T_DS, FOR$INQUIRE,
          FOR$TIME_T_DS, FSCHED, IDTB, MTH$DEXP, MTH$DSORT, OUTDAT,
          PAGEFD, PTITL3

refby: FINDS, NAV

comm: AGFP, AGMP, ARFDIP, ARFP, ARMP, ASFP, ASMP, DCIDEI, DETSIG,
          DETXBI, EARTH, FILNAM, FILTIC, FILTRT, FLTCTL, FTITL1, GRFDIP,
          GRFP, GRMP, HEALCM, IEST, INITVL, LAFP, LAMP, LINAC2, LNAV1,
          MAIN1, MAIN2, MCONCO, MLSFP, MLSMP, MULTDT, NAMES, PLOTS,
          ORAND, RALFP, RALMP, RGFP, RGMP, RIOUT, RIOUT2, SIGTAU,
          SIGVOR, SIMCOM, SYNC, SYSU1, SYSX1, SYSXBO, WIND, WINDCO

name: FSCHED

func: To determine if a particular sensor (addressed by type and
          replication) is scheduled to fail in this simulation run.
          If FSCHED determines that a failure will occur, it determines:
                   * the time of failure
                   * the failure type (i.e. bias, null, etc.)
                   * the simulated failure magnitude

call: Call FSCHED (IsensT,IsensR,convrt,IfailT,failT,failTY,failm)

args: IsensT- integer in   absolute sensor type (from Table 1

                     in [1])
          IsensR- integer in   replication number of this sensor
          convrt- double in   conversion factor to be applied to the
                  failure level
          IfailT- integer out   failure indication flag where:
                  if IfailT = 0 - no filaures are simulated, and if
                  IfailT > 0 then IfailT is the failure type with:
                          0 = no failures
                          1 = increased noise failure
                          2 = increased bias failure
                          3 = increased scale factor failure
                          4 = hardover failure
                          5 = null failure
                          6 = ramp failure
          failT - double out   failure onset time in simulation
                  seconds
          failTY- double out   failure type (logical) string. Ten
                  character string used to indicate the simulated
                  failure mode - if no failures then failTY = '   '
          failm - double out   simulated failure magnitude (in user
                  units)
refs: CHKFL, FLEVEL
refby: PRNTIC
comm: AGFP, AGMP, ASFP, ASMP, LAFP, LAMP, MLSFP, RALFP, RALMP,
         RGFP, RGMP

name: CHKFL
func: This routine checks for the occurrence of a failure. It
         assumes that a sensor can only fail once.
call: Call CHKFL (IpntTF,mxtyp,mxrow,timeF,failT,Ifail)
args: IpntTF- integer in   row number in timeF to be checked
                 (indicates which sensor is to be considered)
         mxtyp - integer in   maximum number of sensor failure modes
                 simulated (also = col. dimension of timeF)
         mxrow - integer in   row dimension of timeF, i.e. matches
                 dimension statement's row dimension for timeF
         timeF - double in   matrix of failure times. The rows
                 correspond to the sensors, and the col. correspond
                 to the failure more. Therefore, if sensor i fails
                 with a bias failure, timeF(i,2) = the time of failure.

(second col. represents bias failures, see section 3.3
in [1] for more details). If timeF(i,j) >= tstop no
failures will be simulated.

    failT - double out   time of failure determined by
            subroutine CHKFL

    Ifail - double out   failure indicator flag. Ifail = 0 if
            no failure and Ifail = failure type if a failure was
            found. See description on Ifailt in subroutine FSCHEP
            for details.

refs: AHEOR
refbv: FSCHED
comm: SETCOM

name: FLEVEL
func: To determine the simulated failure level - returned in
       function
call: X = FLEVEL (IFtype,index,Fin,Fib,Fisf,Fhard,Framp,convrt)
args: IFtype- integer in   absolute failure type
               (1 <= IFtype <= 6) See FSCHED for details.

       index - integer in   index into the failure level vectors
               (sensor indicator). Each failure level vector is
               dimensioned to be the number of sensor types in that
               sensor group. For example, MLS = 3 (azim,el,rng),
               Ias = 1

       Fin -   double in   increased noise failure levels
               (vector)

       Fib -   double in   increased bias failure levels (vector)
               levels

       Fisf - double in   incresed scale factor failure levels
               (vectors)

       Fhard - double in   hardover failure levels (vector)
       Framp - double in   ramp failure levels (vector)
       convrt- double in   conversion factor to be applied to the
               failure level (conversion from program to user units)

       Flevel- double out   simulated failure level in user units
refs: ALTYP, CONVRF, MTH$DSORT
refby: FSCHED
comm: ANGLES, AZELRN, CRTE, DETXBI, EGUIDE, EKF1, GSLOPE,
       INOU, LOGIC4, MCONCO, PSIR, SIMCOM, UPDAT, VARLON

name: OUTDAT
func: To print a double precision array in a formatted fashion.
        Specifically OUTDAT performs the following:
                1) prints a one line comment (up to 70 characters)
                2) if convrt!=1.0 each element of the array is
                   multiplied by convrt
                3) the array is printed with up to 4 (user specified)
                   columns and where each element contains 15 digits.
call: Call OUTDAT (kdsk,A,n,convrt,ncol,Letrs)
args: kdsk - integer in   fortran unit number of the ASCII file
                output will be directed to
        A -   double in   array to be printed
        n -   integer in   length of the array, A
        convrt- double in   conversion factor to be applied to
                all elements of A before printing
        ncol - integer in   desired number of columns (i.e. the
                number of elements per row of printout); 0<ncol<5
        Letrs - char   in   a one line comment which will be
                printed preceeding output of the array
refs: AHEDR
refby: PRNTIC
comm: None

name: TLOUT
func: To print a coded message (corresponding to an "event") in the
        time line (TLN) file. The reader is referred to section 4.2
        on page 71 of [1] for a detailed description of this file and
        its format.
call: Call TLOUT (msg,Imsg1,Imsg2,Imsg3,Imsg4)
args: msg -   integer in   message number corresponding to the
                event # in table 5 in [1]
        Imsg1 - integer in   first message qualifier - corresponds
                to I.D.#1 in table 5 in [1]
        Imsg2 - integer in   second message qualifier - corresponds
                to I.D.#2 in table 5 in [1]
        Imsg3 - integer in   third message qualifier - corresponds
                to I.D.#3 in table 5 in [1]
        Imsg4 - integer in   fourth message qualifier - corresponds
                to I.D.#4 on pages 71-72, where Imsg4=0 means that
                all floating point information will be recorded in

                    absolute values, and Imsg4=1 means they will be
                    recorded as estimation errors
          Note: Information for the floating point descriptors
                    discussed in Section 4.2 in [1] is obtained from the
                    common block variables.
refs: ALTYP, CONVRF, MTH$DSQRT
refby: AIRSPS, ATITGS, AUTLD, BLMAS, BMRGS, CHKRAD, CKUNST, DECIDE,
          FINDS, GETMLS, HEALR, LINAC1, NAV, RADALS, RATEG1, RECONF
comm: ANGLES, AZELRN, CRTE, DETXBI, EGUIDE, EKF1, GSLOPE, INOU,
          LOGIC4, MCONCO, PSIR, SIMCOM, UPDAT, VARLON

3.3.7   Documentation For File: FUTSUB.FOR -

name: ABSLIM
func: Limits an input variable to lie within a symetric bound
        about zero.
call: Call ABSLIM (X,Xlim)
args: X -   double inout   variable to be limited. On return
                  -|Xlim| <= X <= |Xlim|
        Xlim -   double in   value of the boundary
refs: None
refby: AUTOLD, BANKTR
comm: None

name: ACCVEL
func: Computes velocity and acceleration terms. Usually used
        in G-frame.
call: Call ACCVEL (RDg,RDDg,hdot,Vg,dtvg,hddot,psita,xtacc)
args: RDg -   double in   derivative of position vector [3]
        RDDg -   double in   2nd derivative of position vector [3]
        hdot -   double out   vertical component of velocity vector,
                  i.e. hdot = -RDg(3)
        Vg -   double out   magnitude of velocity in x-y plane
        dtvg -   double out   down track velocity
        hddot -   double out   vertical component of acceleration
                  vector, i.e. hddot = -RDDg(3)
        psita -   double out   direction of velocity vector in x-y
                  plane
        xtacc -   double out   cross track acceleration
refs: DATAN2, DSQRT
refby: FINDS
comm: None

name: ROTATV
func: Rotates position, velocity, and acceleration vectors in the
         I-frame into the E-frame and G-frames.
call: Call ROTATV (Ri,RDi,RDDi,comet,sinet,we,wee,rmag,REi,RDe,
         RDg,RDDe,RDDg)
args: Ri -   double in   inertial position vector in the
                  I-frame [3]
         RDi -   double in   inertial velocity vector in the
                  I-frame [3]
         RDDi -   double in   inertial acceleration vector in the
                  I-frame [3]
         comet - double in   cosine of angle swept by the earth's
                  rotation
         sinet - double in   sine of angle swept by the earth's
                  rotation
         we -   double in   earth's rotation rate
         wee -   double in   we * we
         rmag -   double out   absolute value of the length of Ri
         REi -   double out   position vector in the E-frame
         RDe -   double out   earth velocity vector coordinitized in
                  the I-frame
         RDg -   double out   velocity vectors in G-frame
         RDDe -   double out   relative accleration between E and I
                  frames
         RDDg -   double out   RDDe transformed to G-frame
refs: ASIN, DATAN2, MATMUL, ROTMAT
refby: FINDS
comm: TRANS

name: ROTMAT
func: Computes various frame transformation matrices. Common
         block TRANS provides the inputs to this routine and common
         blocks TRANS and ANGS store the results. In particular if.
                  i=1, ANG(1)=phi, ANG(2)=theta and ANG(3)=psi => Tbg
                  i=2, ANG(4)=latitude, ANG(5)=longitude => Til,Tel,Tge
                  i=3, ANG(3)=lat1, ANG(4)=lat2, ANG(5)=lon2-lon1 => Tgg
                  i=4, ANG(1)=phi, ANG(2)=theta, ANG(3)=psi => Tbi
call: Call ROTMAT (i)
args: i -   integer in   flag indicating which transformations
                  to compute

93

refs: DCOS, DSIN, MATMUL
refby: ACEQIN, AUTLD, FINDS, ROTATV, RUNWAY, WAYPNT
comm: ANGS, TRANS

name: RUNGK3
func: Perform Runge-kutta integration for one simulation step
         ahead
call: Call RUNGK3 (dtime,Dx,X,DERSUB,n)
args: dtime - double   in   simulation step size in seconds
         Dx -   double   out   perturbation in X
         X -   double   out   state vector of length n
         DERSUB- double   in   subroutine name of the function to
                  be integrated.   (must be of the form DERSUB(Dx,X,n))
         n -   integer in   length of X and Dx
refs: DERSUB
refby: FINDS
comm: RUNGEK

name: RUNWAY
func: Computes the aircrafts position and velocity vectors in
         the G-frame
call: Call RUNWAY (Reor,Ri,RDi,we,cospsi,sinpsi,cwt,swt,Posit,Veloc)
args: Reor -   double   out   runway origin in the I-frame
         Ri -   double   in   inertial position vector in I-frame
         RDi -   double   in   inertial velocity vector in I-frame
         we -   double   in   earths rotation rate
         cospsi- double   in   cosine of the runway yaw angle
         sinpsi- double   in   sine of the runway yaw angle
         cwt -   double   in   cosine of we*dtime
         swt -   double   in   sine of we*dtime
         Posit - double   out   A/C position in the G-frame
         Veloc - double   out   A/C velocity in the G-frame
refs: ROTMAT, MATMUL
refby: FINDS
comm: TRANS

name: SETUM
func: Initializes all elements of a vector to a constant
        scalar, i.e.
                X(i) = v ; for 1 <= i <= k
call: Call SETUM (X,k,V)
args: X -   double out   vectors to be initialized
        k -   integer in   length of vector X
        V -   double in   value to use for initialization
refs: None
refby: AUTLD
comm: None

name: VECM
func: Multiplies two vectors in an element by element fashion. s.t.
                A(i) = A(i)*B(i) ; for 1 <= i <= n
call: Call VECM (A,B,n)
args: A -   double inout   output vector of length n
        B -   double in   input vector of length n
        n -   integer in   length of A and B
refs: None
refby: BMLAS, BMRGS
comm: None

name: VECMS
func: Increments a vector by the element by element product of
        two other vectors. s.t.
                A(i) = A(i) + B(i)*C(i) ; for 1 <= i <= n
call: Call VECMS (A,B,C,n)
args: A -   double inout   output vector of length n
        B -   double in   input vector of length n
        C -   double in   input vector of length n
        n -   integer in   length of A, B, and C
refs: None
refby: ATITGS, BMLAS, BMRGS, LINAC1, RATEG1
comm: None

```
name:    VECSUM
func:    Increments a vector by another s.t.
                 A(i) = A(i) + B(i)  ;  for 1 <= i <= n
call:    Call VECSUM (A,B,n)
args:    A -      double  inout    vector to be incremented
         B -      double  in       input vector
         n -      integer in       length of A and B
refs:    None
refby:   ATITGS, BMLAS, BMRGS, LINAC1, RATEG1
comm:    None
```


```
name:    MATV3
func:    Multiply a 3x3 matrix by a vector s.t.
                 X = A*Y
         Note:  X and Y CANNOT reside in the same memory locations.
call:    Call MATV3 (X,A,Y)
args:    X -      double  inout    output vector
         A -      double  in       input matrix
         Y -      double  in       input vector
refs:    None
refby:   BMLAS, BMRGS, GTOI
comm:    None
```


```
xDname: MATTV3
func:    Multiply the transpose of a 3x3 matrix by a vector s.t.
                 X = A'*Y
         Note:  X and Y CANNOT reside in the same memory locations.
call:    Call MATTV3 (X,A,Y)
args:    X -      double  inout    output vector
         A -      double  in       input matrix
         Y -      double  in       input vector
refs:    None
refby:   GYROCR
comm:    None
```

name: MATMUL
func: Multiply a 3x3 matrix by a vector (passed as 3 scalar
         elements). Used primarily to multiply the frame
         transformation matrices stored in common block TRANS
call: Call MATMUL (V,a,b,c,d,e,f)
args: V -   double in   matrix stored with rows packed into
                                   a 9 element linear array
         a,b,c - double in   elements of vector multiplied by V
         d,e,f - double out   elements of resultant vector
refs: None
refby: ACEQIN, AUTLD, FINDS, ROTATV, ROTMAT, RUNWAY, WAYPNT
comm: None

name: MOVUM
func: Equates two vectors, i.e.
                   TO = FROM
call: Call MOVUM (FROM,TO,num)
args: FROM -   double in   input array
         TO -   double out   output array
         num -   integer in   length of TO and FROM
refs: None
refby: AUTLD
comm: None

name: DGATIO
func: Prints a matrix out on unit kout with an identifier label
call: Call DGATIO (A,nr,nc,let)
args: A -   double in   matrix to be printed
         nr -   integer in   number of rows in A
         nc -   integer in   number of columns in A
         let -   integer in   4 character name for the matrix
refs: None
refby: BIASF
comm: INOU, MAIN1

97

name: SUMMER
func: Computes the average sum of the elements of a vector.
        Elements are included in the average ONLY if a corresponding
        entry in the row of an index matrix is exactly one.

$$SUMMER = \frac{1}{n} SUM \{X(i)\} \quad ; \quad for\ 1 <= i <= nx,\ and$$

        Index(i). Where n is defined as the number of unit entries
        in Index
call: XX = SUMMER (X,nx,Index)
args: X -   double in   vector to be averaged
        nx -   integer in   length of X and Index
        Index - integer in   row vector (of length nx) whose
                         elements indicate whether corresponding entries in
                         another vector are valid (Index(i)=1) or not
                         (Index(i)!=1). Note it is implicitly assumed that
                         Index is a matrix with row dimension equal to ndim.
        SUMMER- double out   value of the average sum of X
                         conditioned on the elements of Index
refs: None
refby: AVBIAS, SUMIN, SUMOUT
comm: MAIN1

name: ASUMER
func: Computes the average sum of the elements of a vector.
        Elements are included in the average ONLY if the
        absolute value of a corresponding entry in the row of
        an index matrix is exactly one.

$$ASUMMER = \frac{1}{n} SUM \{X(i)\} \quad ; \quad for\ 1 <= i <= nx,\ and$$

        |Index(i)|=1. Where n is defined as the number of unity
        magnitude entries in Index
call: XX = ASUMER (X,nx,Index)
args: X -   double in   vector to be averaged
        nx -   integer in   length of X and Index
        Index - integer in   row vector (of length nx) whose
                         elements indicate whether corresponding entries in
                         another vector are valid (|Index(i)|=1) or not
                         (|Index(i)|!=1). Note it is implicitly assumed that

Index is a matrix with row dimension equal to ndim.

ASUMER- double out   value of the average sum of X
conditioned on the elements of Index

refs: None
refbv: GTOI
comm: MAIN1

name: MAXMIN
func: Searches a double precision vector and determines the
maximum and minimum values and their corresponding
locations.
call: Call MAXMIN (V,npts,vmax.vmin.nmax.nmin)
args: V -   double in   vector to be searched
npts - integer in   length of V (i.e. number of elements
in V to be searched)
vmax - double out   value of the maximum element in V
vmin - double out   value of the minimum element in V
nmax - integer out   location of the maximum element in V
nmin - integer out   location of the minimum element in V
refs: None
refbv: None
comm: None

name: MAXMINS
func: Searches a single precision vector and determines the
maximum and minimum values and their corresponding
locations.
call: Call MAXMINS (V.npts,vmax,vmin,nmax,nmin)
args: V -   real   in   vector to be searched
npts - integer in   length of V (i.e. number of elements
in V to be searched)
vmax - real   out   value of the maximum element in V
vmin - real   out   value of the minimum element in V
nmax - integer out   location of the maximum element in V
nmin - integer out   location of the minimum element in V
refs: None

99

refbv: None
comm: None

name: MXMN2
func: Searches a double precision vector and determines the
         maximum and minimum values and their corresponding
         locations conditioned on the value of a corresponding
         active/inactive flag in a second Vector. Only those
         elements which correspond to "active" elements in the
         conditioning vector are considered in the max-min operation.
call: Call MXMN2 (Imactv,V,npts,vmax,vmin,nmax,nmin)
args: Imactv- integer in   an array of active/inactive flags -
                  s.t. if an element of Imactv is non-zero then a
                  corresponding element in V is active and should be
                  considered in the operation.
         V -   double in   vector to be searched conditioned on
                  Imactv
         npts -   integer in   length of V & Imactv
         vmax -   double out   value of the maximum element in V
         vmin -   double out   value of the minimum element in V
         nmax -   integer out   location of the maximum element in V
         nmin -   integer out   location of the minimum element in V
refs: None
refbv: None
comm: None

name: VECHG1
func: To collapse or expand the size and ordering of a vector. X.
         as directed by a pointer vector. KX. and a flag kflag, s.t.
         The pointer vector KX is simply an array of
         monotonically increasing index pointers into X if kflag=1,
         or Y if Klag=2, which define the proper elements of the
         resulting vector.
                  Y = collapsed X if kflag = 1
                  Y = expanded X if kflag = 2 (new elements are zeroed)
         One of the key features of this routine is that X and Y

100

can be equivalent.

call: Call VECHG1 (kflag,X,KX,Y,n,nmax)

args: kflag - integer in flag indicating to collapse if 1 and
to expand if 2

X - double in input vector to be collapsed/
expanded

KX - integer in pointer vector used to allocate the
proper elements to use in the operation

Y - double out output vector which is
collapsed/expanded version of X

n - integer in the dimension of KX

nmax - integer in the dimensioned length of X and Y.
Note: if expanding X, elements of Y are zeroed out
between n and nmax

refs: None
refby: SAVIT
comm: None

name: MATCG2

func: To add or delete a row in a double precision matrix or vector.
or to add or delete a column in a matrix. If a row or column
is added. its elements are set to zero.

call: Call MATCG2 (jflag,index,Y,nr,nc)

args: jflag - integer in operation flag where:
jflag = 1 add a row : -1 delete a row
jflag = 2 add a column : -2 delete a column

index - integer in pointer to row or column to be added
or deleted

Y - double inout matrix whose index 'th row or column
is to be added or deleted

nr - integer inout number of rows of Y (incremented or
decremented accordingly in MATCG2)

nc - integer inout number of columns of Y (incremented or
decremented accordingly in MATCG2)

refs: ALTYP
refby: CLPSBE. CLPSIO, RECONF
comm: MAIN1

101

name: IMTCG2
func: To add or delete a row in an integer matrix or vector,
          or to add or delete a column in a matrix. If a row or column
          is added, its elements are set to zero.
call: Call IMTCG2 (jflag,index,IY,nr,nc)
args: jflag - integer in   operation flag where:
                  jflag = 1   add a row   ; -1 delete a row
                  jflag = 2   add a column : -2 delete a column
          index - integer in   pointer to row or column to be added
                  or deleted
          IY -   integer inout   matrix whose "index" row or column
                  is to be added or deleted
          nr -   integer inout   number of rows of Y (incremented or
                  decremented accordingly in IMTCG2)
          nc -   integer inout   number of columns of Y (incremented or
                  decremented accordingly in IMTCG2)
refs: ALTYP
refbv: CLPSBE, CLPSIO, RECONF
comm: MAIN1

name: PNTINV
func: Searches a pointer vector for particular entry. The pointer
          vector is an integer array with monotonically increasing
          elements. Typically, a pointer vector will show how a
          (possibly collapsed) vector's elements relate to a standard
          (absolutely indexed) vector. Therefore, this routine can
          be used to answer the following question: "What element of
          the measurement vector (a possibly collapsed vector)
          corresponds to the indicated airspeed's output (an absolute
          index)?"
call: Call PNTINV (isns,Ipoint,n,index)
args: isns - integer in   valve searched for in Ipoint (usually
                  relates to an absolute index in a standard mapping)
          Ipoint - integer in   pointer vector to be searched
          n -   integer in   length of Ipoint
          index - integer out   index in Ipoint where isns was found.
                  If isns was not found index < 0
refs: None
refbv: ADJTBP, CLPSIO, RECONF
comm: None

name: LIMVAL
func: Applies a two sided. symetric limiter about zero to the
          elements of a vector, A. s.t.
                    A(i) = A(i),   if |A(i)| <= BLim(i)
                    A(i) = sign(A(i))*BLim(i),   if |A(i)| > BLim(i)
          where BLim is a vector of absolute limit stops - one for each
          element in A.
call: Call LIMVAL (A,BLim,n)
args: A -   double inout   vector to be limited
          BLim -   double in   vector of absolute limit stops
          n -   integer in   length of A and BLim
refs: None
refby: ATITGS, BMLAS, BMRGS, LINAC1, RATEG1, UPDQ
comm: None

name: LIMVL2
func: Applies a two sided anti-svmetric limiter to the
          elements of a vector. A. s.t.
                    A(i) = BLim(k)   ; if A(i) > BLim(k) - upper limit
                    A(i) = BLim(k+1) ; if A(i) < BLim(k+1) - lower limit
                    A(i) = A(i)   : otherwise
          where k = (i-1)*2+1
call: Call LIMVL2 (A,BLim,n)
args: A -   double inout   vector to be limited
          BLim -   double in   vector of upper and lower limits
                   (2 for each element of A)
          n -   integer in   length of A. and half the length of
                   BLim
refs: None
refbv: AIRSPS
comm: None

name: NOISEG
func: Generates a vector of random samples from a normal
         distribution with zero-mean, and unity variance.
call: Call NOISEG (X,jseed,n)
args: X -   double out   vector of n samples from a N(0,1)
                           gaussian distribution
         jseed - integer inout   seed value for the random number
                           generator
         n -   integer in   length of X
refs: GAUSS
refby: ATTIGS, BMLAS, BMRGS, LINAC1, RATEG1, STARTF
comm: None

name: BARN1
func: Generates a single random sample from either:
                 a N(0,1) distribution if iflag < 0 or,
                 a Uniform(-1,1) distribution if iflag > 0.
call: X = BARN1 (iflag,ikey,iseed)
args: iflag - integer in   flag which determines the distribution
                           from which to select the sample. If iflag < 0 use an
                           N(0,1) distribution, else use a uniform (-1,1)
                           distribution
         ikey -   integer in   not used
         iseed - integer inout   seed value for the random number
                           generator
         BARN1 - double out   the value of the sample conditioned on
                           iflag
refs: UNIFRM, GAUSS
refby: AIRSPS, ATITGS, BMLAS, BMRGS, GETMLS, ILNAG1, IRATG1,
         NOISEG, RADALS
comm: None

name: GAUSS
func: Selects a single random sample from a N(am,s) distribution.
          Where am = mean, and s = standard deviaiton. Note: this
          routine is specific to the VAX computer.
call: Call GAUSS (iseed,s,am,v)
args: iseed - integer inout   seed value for the random number
                  generator
          s -   double in   standard deviation of the distribution
          am -   double in   mean value
          v -   double out   the sample selected
refs: RAN, DSORT, DLOG
refby: BARN1, WINDGT
comm: None

name: UNIFRM
func: Selects a random sample from a uniform distribution
          between 0 and 1. This routine is specific to the VAX
          computer.
call: Call UNIFRM (iseed,v)
ars: iseed - integer inout   seed value for the random number
                  generator
          v -   double out   value of the sample obtained from a
                  Uniform (0,1) distribution.
refs: RAN
refby: BARN1
comm: None

name: NAMFIL
func: To create VAX VMS file names which have a common name, and
          various extensions. The common file name is prompted for
          in the first call to NAMFIL - it can be read from the TTY
          or from a data file.
call: Call NAMFIL (kunit,Lext,Name)
args: kunit - integer in   Fortran unit number from which
                  respones are to be accepted
          Lext - char   in   a 4 character file extension of the

      form ".FOO", which is to be appended to the common
      group name
    Name - char out The resulting (12 character max) file
      name created by concatenating a common group name with
      the specific file extension
refs: ALTYPO, ENCODE
refby: FINDS
comm: None

3.3.8 Documentation For File: FVMSUB.FOR -

name: GMINV
func: Computes the inverse of a square matrix A. If A is
    singular or if A is NOT square, the routine computes
    the Penrose generalized inverse. See Rust, B., Burrus,
    W.R., and Schneeberger, C., "A Simple Algorithm for
    Computing the Generalized Inverse of a Matrix". CACM,
    Vol. 9, No. 5, May 1966.
call: Call GMINV (nr,nc,A,V,mr,mt)
args: nr - integer in number of rows in A
    nc - integer in number of columns in A
    A - double in matrix to be inverted
    U - double out generalized inverse of A
    mr - integer out rank of A
    mt - integer in used for print control, mt=0
      suppresses possible error message printout.
refs: DOT, DOT2, SWAP, VADD
refby: BIASF, DETECT, EKFN1
comm: INOU, MAIN1

```
name:   MMUL
func:   Forms the matrix product
                Z=X Y
        A sparseness test is performed on X.
call:   Call MMUL (X,Y,n1,n2,n3,Z)
args:   X -     double  in      input matrix (n1 x n2)
        Y -     double  in      input matrix (n2 x n3)
        n1 -    integer in      row dimension of X and Z
        n2 -    integer in      col length of X, row length of Y
        n3 -    integer in      col length of Y and Z
        Z -     double  out     output matrix (n1 x n3)
refs:   VADD1
refbv:  BIASF, BLEND, DETECT, EKFN1
comm:   MAIN1
```

```
name:   MMUL2
func:   Forms the matrix product
                Z=X Y
        A sparseness test is performed on Y.
call:   Call MMUL2 (X,Y,n1,n2,n3,Z)
args:   X -     double  in      input matrix (n1 x n2)
        Y -     double  in      input matrix (n2 x n3)
        n1 -    integer in      row dimension of X and Z
        n2 -    integer in      col. length of X, row length of Y
        n3 -    integer in      col. length of Y and Z
        Z -     double  out     output matrix (n1 x n3)
refs:   VADD
refbv:  BLGAIN, DETECT, EKFN1
comm:   MAIN1
```

```
name:   MAT1
func:   Forms the straightforward matrix product
                Z=X Y
        No sparseness tests are performed.
call:   Call MAT1 (X,Y,n1,n2,n3,Z)
args:   X -     double  in      input matrix (n1 x n2)
```

107

```
            Y -      double   in     input matrix (n2 x n3)
            n1 -     integer  in     row dimension of X and Z
            n2 -     integer  in     col. length of X, row length of Y
            n3 -     integer  in     col. length of Y and Z
            Z -      double   out    output matrix (n1 x n3)
refs:   DOT3
refby:  BIASF, BLEND, BLGAIN, DETECT, MAT3, MAT3A
comm:   MAIN1
```

```
name:   MAT1A
func:   Forms the matrix product
                Z=X Y
        No sparseness tests are performed.  Z and Y can start
        at equivalent core locations.
call:   Call MAT1A (X,Y,n1,n2,n3,Z)
args:   X -      double   in     input matrix (n1 x n2)
        Y -      double   in     input matrix (n2 x n3)
        n1 -     integer  in     row dimension of X and Z
        n2 -     integer  in     col. length of X, row length of Y
        n3 -     integer  in     col. length of Y and Z
        Z -      double   out    output matrix (n1 x n3)
refs:   None
refby:  BIASF, EKFN1
comm:   MAIN1
```

```
name:   MAT2
func:   Forms the matrix product
                Z=XY'
        in cases where the product Z is SYMMETRIC.  No
        sparseness tests are done.  The arrays Z and Y
        can start at equivalent core locations.
call:   Call MAT2 (n1,n2,X,Y,Z)
args:   n1 -     integer  in     row dimension of X,Y, and col.
                                 length of Z.
        n2 -     integer  in     col. dimension of X and Y
        X -      double   in     input matrix (n1 x n2)
```

```
           Y -      double   in      input matrix (n1 x n2)
           Z -      double   out     output matrix (n1 x n2)
refs:    DOT2
refbv:   EKFN1
comm:    MAIN1
```

```
name:    MAT3
func:    Forms the symmetric matrix product
               Z = X Y X'
         where Y is symmetric, and no sparseness tests are done.
call:    Call MAT3 (n1,n2,X,Y,Z)
args:    n1 -     integer in      row length of X and Z, col.
                  length of Z.
         n2 -     integer in      col. length of X and Y, row
                  length of Y
         X -      double   in      input matrix (n1 x n2)
         Y -      double   in      input (symmetric) matrix (n2 x n2)
         Z -      double   in      output (symmetric) matrix (n1 x n1)
refs:    DOT2, MAT1
refbv:   BIASF, EKFN1
comm:    MAIN1
```

```
name:    MAT3A
func:    Forms the symmetric matrix product
               Z = X Y X
         where Y is symmetric, and no sparseness tests are done.
call:    Call MAT3A (n1,n2,X,Y,Z)
args:    n1 -     integer in      row length of Z, col. length of Z and
X
         n2 -     integer in      col. length of Y, row length of Y and
X
         X -      double   in      input matrix (n2 x n1)
         Y -      double   in      input (symmetric) matrix (n2 x n2)
         Z -      double   in      output (symmetric) matrix (n1 x n1)
refs:    DOT, MAT1
refbv:   BIASF, EKF1
```

comm: MAIN1

name: MAT4
func: Forms the matrix product
                  Z = XY'
          No sparseness tests are performed.
call: Call MAT4 (X,Y,n1,n2,n3,Z)
args: X -   double in   input matrix (n1 x n2)
          Y -   double in   input matrix (n3 x n2)
          n1 -   integer in   row dimension of X and Z
          n2 -   integer in   col. length of X and Y
          n3 -   integer in   row length of Y, col. length of Z
          Z -   double out   output matrix (n1 x n3)
refs: DOT2
refby: BIASF, BLEND
comm: MAIN1

name: MAT5
func: Forms the matrix product
                  Z = XY'
          A sparseness test is performed on Y.
call: Call MAT5 (X,Y,n1,n2,n3,Z)
args: X -   double in   input matrix (n1 x n2)
          Y -   double in   input matrix (n3 x n2)
          n1 -   integer in   row dimension of X and Z
          n2 -   integer in   col. length of X and Y
          n3 -   integer in   row length of Y, col. length of Z
          Z -   double out   output matrix (n1 x n3)
refs: VADD, VSCALE
refby: EKFN1
comm: MAIN1

name: MAT6
func: Forms the matrix product
                Z = XY'
        in cases where Z is symmetric. A sparseness test is
        performed on Y. Neither X nor Y may be equivalent to Z.
call: Call MAT6 (n1,n2,X,Y,Z)
args: n1 -   integer in   row length of X,Y, and Z
        n2 -   integer in   col. length of X, Y, and Z
        X -   double in   input matrix (n1 x n2)
        Y -   double in   input (symmetric) matrix (n1 x n2)
        Z -   double out   output (symmetric) matrix (n1 x n1)
refs: VADD
refby: DETECT
comm: MAIN1

name: MADD1
func: Adds two matrices as follows:
                Z = X+c1*Y
call: Call MADD1 (nr,nc,X,Y,Z,c1)
args: nr -   integer in   row length of X,Y, and Z
        nc -   integer in   col. length of X,Y, and Z
        X -   double in   input matrix (nr x nc)
        Y -   double in   input matrix (nr x nc)
        Z -   double out   output matrix (nr x nc)
        c1 -   double in   scale factor applied to Y
refs: None
refby: BIASF, BLEND, BLGAIN, DETECT, EKFN1
comm: MAIN1

name: MADDI
func: Sets up square matrix A where:
                A = c1*B+c2*I
        I is an identity matrix.
call: Call MADDI (n,A,B,c1,c2)
args: n -   integer in   size of matrices
        A -   double out   output matrix (n x n)

```
            B -      double  in      input matrix (n x n)
            c1 -     double  in      scale factor applied to B
            c2 -     double  in      scale factor applied to I
refs:       None
refby:      BIASF, BLGAIN, DETECT, EKFN1
comm:       MAIN1
```

```
name:       EQUATE
func:       Sets a matrix A equal to a matrix B (can be used for
            equating matrix partitions or sub-blocks as well)
                  A=B
call:       Call EQUATE (A,B,nr,nc)
args:       A -      double  out     output matrix (nr x nc)
            B -      double  in      input matrix (nr x nc)
            nr -     integer in      row length of A and B
            nc -     integer in      col. length of A and B
refs:       None
refby:      BIASF, BLEND, DETECT, EKFN1, FILCOL, INITG, RECONF
comm:       MAIN1
```

```
name:       MATNUL
func:       Initializes columns of a matrix to zero.  Where:
                  Xi = 0, for   n1 <= i <= n2:
            and Xi is the ith col. of X.  In addition, if a flag is
            set, rows between n1 and n2 can be nulled out as well.
call:       Call MATNUL (X,n1,n2,ktrig)
args:       X -      double  inout   matrix to be nulled
            n1 -     integer in      first col. (row) to be nulled
            n2 -     integer in      last col. (row) to be nulled
            ktrig -  integer in      flag, when ktrig=0 only columns
                     are nulled, otherwise rows and columns are nulled
refs:       None
refby:      DETECT, RCOV, RECONF
comm:       MAIN1
```

name: MSCALE
func: Sets a matrix A equal to a matrix B and scales.
        $A = c1*B$
call: Call MSCALE (A,B,nr,nc,c1)
args: A -   double out   output matrix (nr x nc)
        B -   double in   input matrix (nr x nc)
        nr -   integer in   row length of A and B
        nc -   integer in   col. length of A and B
        c1 -   double in   scale factor applied to B
refs: None
refby: BLGAIN, DETECT, EKFN1, RECONF, UPDPH
comm: MAIN1

name: TRANS2
func: Transpose a matrix
                $AT = A'$
call: Call TRANS2 (n1,n2,A,AT)
args: n1 -   integer in   row length of A, col. length of AT
        n2 -   integer in   col. length of A, row length of AT
        A -   double in   matrix to be transposed n1 x n2)
        AT -   double out   transposed matrix (n2 x n1)
refs: None
refby: BLEND
comm: MAIN1

name: BUBBLE
func: Performs a bubble sort on an array of integers. The
        elements of the array can be ordered increasing or
        decreasing in value.
call: Call BUBBLE (NA,n,k)
args: NA -   integer inout   array of integers to be sorted
        n -   integer in   length of array NA
        k -   integer in   a key, where k>0 orders NA decreasing
                                in value while k<=0 yields an increasing order
refs: None
refby: HEALR, INITG, READRC

113

comm: None

name: DOT
func: Computes the dot (or inner) product between two linear
        arrays (column vectors), with accumulation carried out
        in double precision.
call: x = DOT (nr,A,B)
args: nr -   integer in   length of arrays A and B
        A -   double in   vector
        B -   double in   vector
refs: None
refby: GMINV, MAT3A
comm: None

name: DOT2
func: Computes the dot (or inner) product between two rows of
        a matrix.
call: x = DOT2 (nn,A,B)
args: nn -   integer in   length of A(B) times the dimensioned
                row length of A(B)
        A -   double in   row vector (or row of a matrix)
        B -   double in   row vector
refs: None
refby: GMINV, MAT2, MAT3, MAT4
comm: MAIN1   ndim - dimensioned row length of A and B

name: DOT3
func: Computes the dot (or inner) product between two arrays,
        where one array is stored as a row vector and the other
        as a column vector.
call: x = DOT3 (n,A,B)
args: n -   integer in   length of A and B

114

	A -	double	in	row vector
	B -	double	in	column vetor
refs:	None			
refby:	MAT1			
comm:	MAIN1	ndim -	dimensioned row length of A	

name: VADD
func: To increment a given vector A by a second vector s.t.:
                A = A+c1*B
call: Call VADD (n,c1,A,B)
args: n -   integer in   length of A and B
        c1 -   double in   scale factor
        A -   double inout vector to be incremented
        B -   double in   vector to scale with
refs: None
refby: GMINV, MAT5, MAT6, MMUL2
comm: None

name: VADD1
func: To increment a given row vector A by a second row vector
        B s.t.:
                Arow = Arow+c1*Brow
        This routine assumes A and B are stored as matrices.
call: Call VADD1 (nn,c1,A,B)
args: nn -   integer in   length of A(B) times the dimensioned
                 row length of A(B)
        c1 -   double in   scale factor
        A -   double inout row vector to be incremented
        B -   double in   row vector to be scale
refs: None
refby: MMUL
comm: MAIN1   ndim - dimensioned row length of A and B

115

name: VSCALE
func: Equates a vector A to a scaled vector B. A and B can be
        equivalent.
                A = c1*B
call: Call VSCALE (A,B,n,c1)
args: A -   double out   vector to store result in
        B -   double in   vector to be scaled
        n -   integer in   length of A and B
        c1 -   integer in   scale factor
refs: None
refby: MAT5
comm: None

name: SEQNCE
func: Initializes an integer array to a monotonically
        increasing sequence s.t.:
                K = [1,2,3,...,n,0,0,...,0]'
call: Call SEQNCE (K,n)
args: K -   integer out   array to be initialized
        n -   integer in   length of sequence to be stored in K
refs: None
refby: INITG. READRC. SAVIT
comm: MAIN1 - ndim - dimensioned length of K

name: INSRTN
func: Used to update (and maintain) an integer vector (of
        pointers) with a new (unique) value. The new value
        is added to the list ONLY if:
                1. it's not already present in the list
                2. the current length of the list is >=0
                3. the current length of the list is < a
                    maximum length.
call: Call INSRTN (Iseq. index, ivalue, mxsize)
args: Iseq - integer inout   array (list) of unique integers
        index - integer inout   current length of the list
        ivalue- integer in   candidate for addition to the list.

```
                Iseq
        mxsize- integer in       maximum (dimensioned) length of Iseq
refs:   None
refby:  READRC
comm:   None




name:   VECNULS
func:   Initializes a linear array to zero - single precision version.
        Where:
                X(i) = 0.0   for i1 <= i <= i2
call:   Call VECNULS (X,i1,i2)
args:   X -     real    inout   vector to be nulled
        i1 -    integer in      starting element to null
        i2 -    integer in      final element to null
refs:   None
refby:  None
comm:   None




name:   VECNUL
func:   Initializes a linear array to zero - double precision
        version.  Where:
                X(i) = 0.0  for i1 <= i <= i2
call:   Call VECNULS (X,i1,i2)
args:   X -     double  inout   vector to be nulled
        i1 -    integer in      starting element to null
        i2 -    integer in      final element to null
refs:   None
refby:  BLEND, CHKRAD, DETECT, SAVIT
comm:   None
```

117

name: SWAP
func: Interchanges 2 rows. 2 columns, or 2 diagonals of two
        matrices.
call: Call SWAP (A,B,n,inc)
args: A -   double inout   a matrix to be interchanged
        B -   double inout   a matrix to be interchanged
        n -   integer in   number of elements to be swapped
        inc - integer in   interleaving factor. where: inc = 1
               swaps columns. inc = ndim swaps rows, and inc = ndim+1
               swaps diagonals. Where ndim is the row dimension of A
               and B.
refs: None
refby: GMINV
comm: None

name: VMAT1
func: Multiplies a given vector by a matrix.
             Y = A X
        where: X is an n2 vector and A is an n1 by n2 matrix.
call: Call VMAT1 (A,X,n1,n2,Y)
args: A -   double in   input matrix (n1 x n2)
        X -   double in   input vector (n2)
        n1 -   integer in   row of A and length of Y
        n2 -   integer in   col. of A and X
        Y -   double out   output vector (n1)
refs: None
refby: BLEND
comm: MAIN1   ndim - dimensioned row length of A

name: VMAT2
func: Computes the vector-matrix product sum
             Y = Z+A X
        where is an n2 vector. and A is an n1 by n2 matrix. Y
        and Z can be equivalent.
call: Call VMAT2 (Z,A,X,n1,n2,Y)
args: Z -   double inout   input vector (n1)

	A –	double	in	input matrix (n1 x n2)
	X –	double	in	input vector (n2)
	n1 –	integer	in	row length of A, col. length of
		Y and Z		
	n2 –	integer	in	col. length of A and X
	Y –	double	out	output vector (n1)
refs:	None			
refby:	BLEND			
comm:	MAIN1	ndim – dimensioned row length of A

3.3.9   Documentation For File: PLOTD.FOR –

name: PLOTD
func: Utility program to plot the unformatted (binary) time history
        data stored in the .PLT file generated by FINDS.
call: To invoke PLOTD, the user simply types (at the VMS monitor
        level):
                $ RUN PLOTD/NODEBUG
        or, if FOREIGN.COM has been executed :
                $ PLOTD
        PLOTD will then prompt the user for various directive
        commands. Time history data is identified by a unique name
        stored in the header of the .PLT file. (See Table 6 on
        page 82 of [1] for a current list of these names.)
outs: PLOTD can be used to generate plots of one or several
        variables versus time, or to create cross plots of one
        variable vrs. another. Currently PLOTD creates a single plot
        per page. The plots can be generated on a TEKTRONIX
        4010/4014 or any terminal capable of emulating a 4010 or 4014.

3.3.10   Documentation For File: PRINTD.FOR -

name: PRINTD
func: Utility program used to examine the unformatted (binary) time
        history data stored in the .PLT file generated by FINDS.
        PRINTD can be used to either display selected data in tabular
        form, or to compute temporal means and autocorrelations. The
        results are presented to the users terminal, the system line
        printer, or to a user specified data file.
call: To invoke PRINTD, the user simply types (at the VMS monitor
        level):
                $ RUN PRINTD/NODEBUG
        or, if FOREIGN.COM has been executed :
                $ PRINTD
        PRINTD will then prompt the user for various directive
        commands. Time history data is identified by a unique name
        stored in the header of the .PLT file. (See Table 6 on
        page 82 of [1] for a current list of these names.)
outs: either:
        a. A table of selected data, where each column of data is
           headed by the name and engineering units. Data can be
           "windowed" by selecting upper and lower temporal limits.
           Within a window, one can further segment the data by
           specification of a constant skip factor.
        b. All of the following:
           1. The sample mean.
           2. The sample variance.
           3. The sample autocorrelation function normalized by the
              sample variance.
           4. The decision of a whiteness test performed on the
              selected data.
        Please see Appendix A.1 of [1] for a more complete
        description.

3.3.11 Documentation For File: DOC.RAT -

name: DOC
        (Ratfor and Fortran Documentation generator)
func: Prepares a RUNOFF input file from specially formatted embedded
        documentation contained in a RATFOR or FORTRAN program or
        group of programs. Each source file is entered in the table
        of contents and each source file, subroutine, and common block
        is entered in the index.
        The program is executed by typing:
                $ DOC
        The user is prompted for the following items:
        | Output file name | The name to use for the RUNOFF input |
        | | file |
        | Header level | A header at this level is created for |
        | | each separate source file, containing |
        | | the source file name. |
        | Line length | Line length to specify to RUNOFF |
        | Input file | Name of source file to be processed. |
        | | or name of indirect file. |

        No default extensions are assumed for any of the above.
        In place of specifying a source file as input, the user may
        specify an indirect file by entering "@indirect_file_name"
        when prompted for an input file. The indirect file should
        contain a list of the source files to be processed. This
        option is useful when processing complicated programs spread
        over many source files.
Installation notes:
        Simply compile and link DOC, then execute this command:
                DOC :== Run/nodebug DOC
Source program formating notes:
        A documentation header must begin with the characters "#{doc",
        or "Cdoc" starting in the first column on a separate line, and
        end with the characters "} #" (without the space), or
        "Cenddoc" also on a seperate line for RATFOR, and FORTRAN
        sources respectively. All the enclosed text will be included
        in the runoff file with the following exceptions:

121

* The comment character "C" is stripped off the first column
  of each line in FORTRAN sources.
* Lines of the form "FIG: FFPF(n)" will be used to set aside
  pages for "Floating Full Page Figures". The number 1< n <99
  is the number of pages required for the figure.
Any data in a line beginning "name:" will be assumed to be the
name of a subroutine or common block and hence will be entered
in the index.

4   INTERNAL DATA STRUCTURES

This section describes the important common blocks used by FINDS routines to communicate with each other. The first section reviews some of the important assumptions and concepts used in building and manipulating the internal data structures. The last section describes each common block in detail. Each such description contains a statement about the contents of the common block and a description of each variable in the form:

> name   type   units   description

These are followed by a list of all the routines which use the common block.

4.1   Data Structure Conventions

In the course of developing FINDS, various indexing schemes were required, as well as many special purpose data storage conventions. Many of these conventions become apparent when the detailed flow diagrams are studied carefully. The individual realizations of these methods are described in the next subsections. This subsection attempts to describe the conventions and concepts themselves.

The general storage format used for matrices is to allocate (dimension) them "ndim" by "ndim", where "ndim" is an integer variable stored in common block /MAIN1/ (ndim=15 in this version of FINDS). The i j element of the matrix is then stored in the i j element of the storage area. Therefore, if the matrix to be stored were of size 5 by 5, and ndim = 15, then we view the matrix as a linear array, with a column offset of 15 elements (i.e. five data locations followed by ten unused elements in each of 5 columns). Although this storage scheme is less efficient (from a memory access point of view) than simply storing the columns contiguously (with a column offset of five), it was necessary in order to use many of the utility routines documented in file FUTIL.FOR.

The origins of most of the internal data storage conventions can be grouped into the following areas:

. No-fail filter
. FDI logic
. Reconfiguration

The first two areas require internal data structures that maintain an absolute index - so the program can relate states, measurements, inputs, failures, etc., to particular "physical" sensors or quantities. The last area, however, imposes a need to modify the absolute ordering to reflect loss or addition of a sensor. Tables 1-7 define the important absolute indexing schemes employed by FINDS. These tables not only define conventions for particular arrays, but also implicitly define all the matrices which operate on them.

As mentioned above, in order for FINDS to be capable of reconfiguring itself the absolute indexing schemes had to be modified. This was accomplished by using two techniques. They are:

. use pointer arrays to provide the mapping between the absolute indexing scheme (actual locations of the data) and the current collapsed/expanded sets.
. physically collapse or expand the arrays

Both methods are used in FINDS. The following is a typical example of the first method:

YF1 is a fixed length vector of averaged measurements presented to the no-fail filter. It uses the absolute measurement indexing convention discussed in Table 2. If ALL replications of a particular type of sensor have failed, and are therefore not available to the NFF, the corresponding element in YF1 is zeroed out. A pointer array INOYP is used to provide the mapping between the (possibly) collapsed measurement vector required by the NFF and the fixed length vector YF1 which is maintained. Figure 11 graphically shows how this arrangement works. The important point to see here is that the data is physically stored in the array using the absolute indexing scheme, and it is extracted using the pointer array INOYP which accounts for any reconfiguration.

An example of the second method is as follows:

RESBOC is a variable length vector of expanded residuals
from the NFF. The vector is formed by first computing the
residuals using the (absolute) replicated measurement
indexing convention (see Table 7), and then collapsing
it to eliminate elements corresponding to sensors which
are not available. The pointer array, INORYP is used to
map each element to the absolute index in Table 7.
Figure 12 shows how this approach works. Notice that
here the data is stored in a collapsed fashion and
INORYP is used to identify each element (the value of
an element in INORYP is the measurement index in Table 7)

The following arrays use this method of organization:

HP1, RESBOG, RESBOC, RBFO, and CBFO

Table 1. No-Fall Filter Absolute State Indexing Conventions

Array Index	State Variable	Program Units
1	Xr_w	feet
2	Yr_w	feet
3	Zr_w	feet
4	$\dot{X}r_w$	feet/sec
5	$\dot{Y}r_w$	feet/sec
6	$\dot{Z}r_w$	feet/sec
7	Phi	radians
8	Theta	radians
9	Psi	radians
10	X_w	feet/sec
11	Y_w	feet/sec

Table 2. No-Fail Filter Absolute Measurement Indexing Conventions

Array Index	Measurement Name	Program Units
1	Azm	radians
2	El	radians
3	Rng	feet
4	IAS	feet/sec
5	Phi	radians
6	Theta	radians
7	Psi	radians
8	RA	feet

Table 3. No-Fail Filter Absolute Input Indexing Conventions

Array Index	Input Name	Program Units
1	Ax	feet/sec/sec
2	Ay	feet/sec/sec
3	Az	feet/sec/sec
4	P	radians/sec
5	Q	radians/sec
6	R	radians/sec

Table 4. No-Fail Filter Process Noise Indexing Conventions

Array Index	Name	Program Units
1	Ax	feet/sec/sec
2	Ay	feet/sec/sec
3	Az	feet/sec/sec
4	P	radians/sec
5	Q	radians/sec
6	R	radians/sec
7	Xw	feet/sec
8	Yw	feet/sec

Table 5. Absolute Sensor Indexing Conventions

Array Index	Sensor Type	Program Units
1	Ax	feet/sec/sec
2	Ay	feet/sec/sec
3	Az	feet/sec/sec
4	P	radians/sec
5	Q	radians/sec
6	R	radians/sec
7	Azm	radians
8	El	radians
9	Rng	feet
10	IAS	feet/sec
11	Phi	radians
12	Theta	radians
13	Psi	radians
14	RA	feet

Table 6. Replicated Sensor Indexing Convention

Array Index	Sensor Type/Replication	Program Units
1	Ax-*	feet/sec/sec
2	Ay-*	feet/sec/sec
3	Az-*	feet/sec/sec
4	P-*	radians/sec
5	Q-*	radians/sec
6	R-*	radians/sec
7	Azm-1	radians
8	El-1	radians
9	Rng-1	feet
10	IAS-1	feet/sec
11	Phi-1	radians
12	Theta-1	radians
13	Psi-1	radians
14	RA-1	feet
15	Azm-2	radians
16	El-2	radians
17	Rng-2	feet
18	IAS-2	feet/sec
19	Phi-2	radians
20	Theta-2	radians
21	Psi-2	radians
22	RA-2	feet

* - refers to the replication currently in use by the NFF (e.g. 1, 2, or 3)

Table 7. Replicated Measurement Indexing Convention

Array Index	Measurement Sensor Type/Replication	Program Units
1	Azm-1	radians
2	El-1	radians
3	Rng-1	feet
4	IAS-1	feet/sec
5	Phi-1	radians
6	Theta-1	radians
7	Psi-1	radians
8	RA-1	feet
9	Azm-2	radians
10	El-2	radians
11	Rng-2	feet
12	IAS-2	feet/sec
13	Phi-2	radians
14	Theta-2	radians
15	Psi-2	radians
16	RA-2	feet

INOYP   YF1

ny   nymax
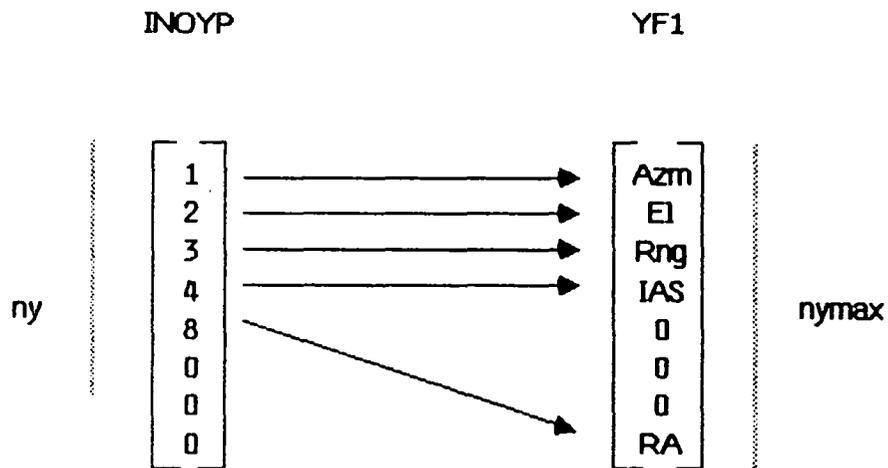
Figure 11. Example of Pointer Array Indexing

RESBOC   INORYP

nyf   nyf
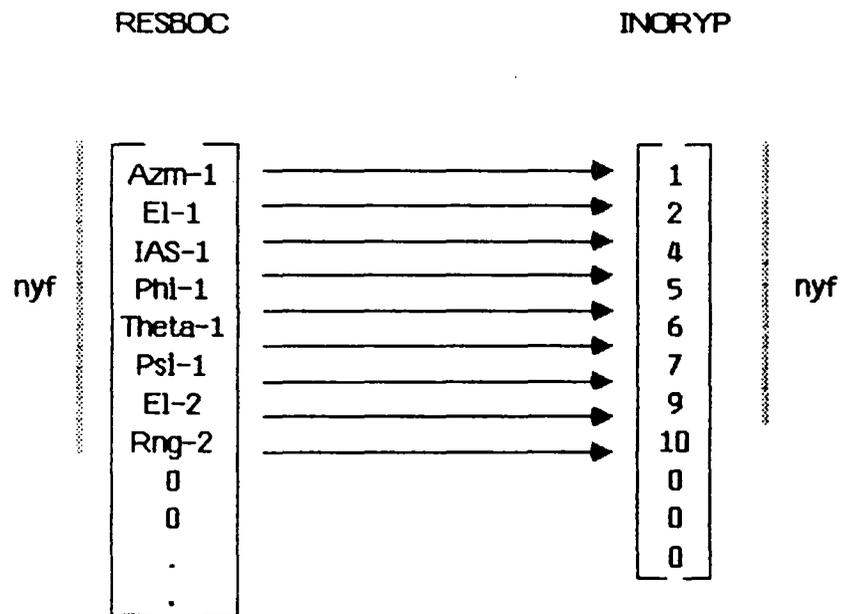
Figure 12. Example of Collapsed Array Indexing

4.2 Detailed Descriptions Of FINDS Common Blocks

4.2.1 Description Of CMPSTF -
name: CMPSTF
cont: Ouantities associated with the composite (bias-free plus bias)
         no-fail filter.
vars: nxb -   integer unitless   the total number of states and
                  bias states in the NFF
         GAINKX- double mixed   the combined no-fail filter
                  qain matrix
         PXF1 -   double mixed   the combined no-fail filter
                  estimation error covariance. (see 2.2.32-2.2.35
                  in [2])
refby: ADJTBP. BIASF, BLEND. CHKRAD, DETECT, INITG, NAV, RCOV,
         SAVIT, STARTF, UPDPH

4.2.2 Description Of DCIDEI -
name: DCIDEI
cont: Ouantities needed bv the LR computations and the decision
         loqic.
vars: Ihyp -   integer unitless   the hvpothesis chosen by the
                  decision loqic. Where Ihvp = the replicated sensor
                  index of the failed sensor if Ihvp<NFT1. or if Ihvp =
                  NFT1 it siqnifies nothinq has failed. whereas if
                  NFT1<Ihyp<NFT1+3 then it siqnifies a multiple failure
                  of MLS azimuth. elevation. or ranqe respectively
         kCTdwh- inteqer unitless   counter for elapsed samples
                  since last decision window was started
         kMXdwh- integer unitless   maximum number of samples in a
                  decision window (i.e. number of samples/decision
                  window)
         kCTewh- integer unitless   vector of decision window
                  counters - one for each detector. The elements
                  are arranqed bv absolute replicated sensor index
         kMXewh- integer unitless   vector of maximum decision

windows in an estimation window (i.e. number of
decision windows/estimation windows) - one for each
detector

PRIORI- double unitless   vector of the log of the prior
probabilities of failure - one for each sensor,
ordered by replicated sensor index

ALamda- double unitless   vector of the log-likelihood
of a sensor failing - one for each sensor, ordered
by replicated sensor index

BetaI - double unitless   not used currently

COSTI - double unitless   not used currently

refby: CHKRAD. DECIDE. DETECT. INITG. NAV. PRNTIC. RECONF. SAVIT

4.2.3   Description Of DETINF -
name: DETINF
cont: Information pertinent to the detectors
vars: nft -   integer unitless   the total number of replicated
sensors (considered for FDI)

nftl -   integer unitless   nft+1

nvf -   integer unitless   the current number of
replicated measurement sensors

INORYP- integer unitless   pointer vector to the
measurement sensor type (from Table 1 in [1]). The
array index is the replicated (and possibly collapsed)
set of sensors used by the NFF, and the value of an
element of INORYP is the absolute sensor type of that
sensor

ICNTSN- integer unitless   ICNTSN is used to determine
  1.   if a particular sensor type and
       replication is being used by the NFF
  2.   and if it is being used - which element of
       the input vector or expanded measurement
       vector it corresponds to

ICNTSN is organized as follows: the array index
corresponds to the absolute replicated sensor index.
the value is either
  1)   the index in the input vector

                            (if index<=NU1)
                  2)   or the index in the expanded measurement
                       vector (if index>NU1)
              if the sensor is not used by the NFF the value of its
              element in ICNTSN is zero
        RESBOC- double mixed   expanded residual vector from
              the no-fail filter. (see 2.3.1-2.3.3 in [2])
refby: BLEND. CLPSIO. DECIDE. DETECT. FILCOL. INITG, NAV. RECONF.
        RESCMP. SAVIT. SETISN

4.2.4   Description Of DETSIG –
name: DETSIG
cont: Sensor noise statistics assumed by the detectors.
vars: PDETCT- double mixed   vector containing standard
                deviations of the expected noise (in program units)
                assumed for each sensor type by the detectors. PDETCT
                is ordered by absolute sensor type
refbv: INITG. PRNTIC

4.2.5   Description Of DETXBI –
name: DETXBI
cont: Quantities associated with the sensor failure detectors
vars: NF –   integer unitless   current number of sensor
                TYPES that are active (i.e. not failed)
        NFmax – integer unitless   maximum possible number of
                sensor TYPES that can be considered
        NYmax – integer unitless   maximum possible number of
                measurement sensor types that can be considered
        XBFI –   double mixed   vector of current failure
                level estimates – one for each detector. Detectors

136

are ordered using the absolute indexing scheme for
replicated sensors. (see 2.3.18 in [2])

PBFI - double mixed   vector of estimation
information for each estimated failure. Ordered in
the same fashion as XBFI. (see 2.3.20 in [2])

VBI - double mixed   matrix of blender gain vectors
where each column of VBI is a blender gain vector.
The columns are indexed using the same scheme as XBFI.
(see 2.3.17 in [2])

BDFI - double mixed matrix of partial derivatives
evaluated about the current failure estimates.
Specifically, it is the partial of BF1 w.r.t. failures
in phi, theta, and psi. The matrix is organized as a
partitioned matrix with each partition of size NX rows
by 3 columns. The partitions are ordered (in the
column direction) as the partial of BF1 w.r.t., the
first replication of phi, theta, psi, the second
replication of phi, theta, psi, and so on. If dual
redundancy is assumed the entire matrix would be
nx by 18. (see 2.3.16 in [2])

refbv: ADJTBP, CHKRAD, CLPSIO, DECIDE, DETECT, INITF, INITG, PRNTIC,
RECONF, SAVIT, SETISN, SUMOUT, TLOUT, UPDAB

4.2.6   Description Of DETYBI -

name: DETYBI
cont: Observation matrices and compensated residual vectors for the
bank of detectors
vars: RESBI - double mixed   matrix of failure compensated
residuals vectors where each column of RESBI is a
residuals vector. The columns are ordered by
replicated sensor index. (see 2.3.14 in [2])

CBFI - double mixed   matrix of detector observation
matrices where each column of CBFI is an observations
vector for a detector. The columns are ordered bv
replicated sensor index. (see 2.3.15 in [2])
refbv: DETECT, INITG, SAVIT

4.2.7   Description Of EKBFO -
name: EKBFO
cont: Bias filter arrays used in the bias fiter portion of the
        no-fail fiter (extended Kalman fiter). (see [4])
vars: XBFO - double mixed   bias filter state vector (i.e.
                vector of current normal operating bias estimates
        RESBO - double mixed   residuals vector generated by
                the bias filter portion of the NFF
        GAINBO- double mixed   Kalman gain for the bias
                filter
        PBFO - double mixed   bias filter estimation
                error covariance (or information)
refbv: BIASF. BLEND. CHKRAD. CLPSBE. CLPSIO. DETECT. INITG. NAV.
        ARCOV. RECONF. RESCMP. SAVIT. SUMIN

4.2.8   Description Of FCOM1 -
name: FCOM1
cont: Communication and common variables between FILER1 and
        RECRDS. All quantities are therefore used in generating
        the binary PLT file.
vars: ntick - integer unitless   ratio of no. of simulation
                steps/record step
        itick - integer unitless   counter variable. when
                itick=ntick variables are recorded in the logical
                record
        nchan - integer unitless   total no. of channels to be
                saved minus one
        jchan - integer unitless   current channel number being
                saved
        mxchan- integer unitless   maximum no. of channels
                allowed
        nbuf - integer unitless   (fixed) length of the physical
                record buffer
        ibuf - integer unitless   current length of the physical
                record
        ifold - integer unitless   flag indicating a previous
                call

138

                    to RECRDS when ipass=2 or 4
            xbuf - real   mixed   array of length nbuf used to
                    store the logical records
refby: FILER1. FINDS. INITG. NAV. RECRDS. SAVIT. SET

4.2.9   Description Of FCOM2 -
name: FCOM2
cont: Storage for the names and units of all variables saved
        in the PLT file
vars: Lname - char   string   list of unique 5 character
                names for each variable stored in the PLT file.
                Lname is of length mxchan
        Lunit - char   string   list of 5 character names for
                the engineering units associated with each variable
                stored in the PLT file. Lname is of length mxchan
refby: FILER1. FINDS. RECRDS

4.2.10 Description Of FILNAM -
name: FILNAM
cont: Stores the (12 character) names of all the disk files used
        by FINDS.
vars: KKBLNK- char   string   blank name used for testing
        FNDK1 - char   string   name of PLT file
        FNDK5 - char   string   name of general input file
        FNDK6 - char   string   name of OUT file
        FNDK7 - char   string   name of filter input file
        FNDK8 - char   string   name of TLN file
        FNDK11- char   string   name of SUM file
        FNDK12- char   string   name of sensor input file
refby: FILER1. FINDS. PRNTIC

139

4.2.11   Description Of FILTIC -
name: FILTIC
cont: Additional initial conditions for the no-fail filter
vars: SDXIC - double mixed   vectors of standard deviations
                 which define the statistics of an initial normal
                 distribution used to choose the initial no-fail filter
                 state estimation error (stored in user units)
         XICerr- double mixed   vector of initial no-fail
                 filter state estimation errors (stored in user units)
         SDPIC - double mixed   vector of standard deviations
                 of the diagonal elements of the no-fail filter state
                 estimation error covariance (stored in user units)
         POSbnd- double feet   position error bound for the
                 no-fail filter's divergence test
         VELbnd- double feet/sec   velocity error bound for the
                 no-fail filter's divergence test
         ANGbnd- double radians   angular error bound for the
                 no-fail filter's divergence test
refby: CKUNST, INITG, STARTF, PRINTIC

4.2.12   Description Of FILTRT -
name: FILTRT
cont: Flags and pointing vectors used by the no-fail filter
vars: Iupc - integer unitless   (not currently used) rate at
                 which NFF covariance is updated [= 1/(dt*Iupc)]
         ikc -   integer unitless   (not currenty used) rate at
                 which NFF is updated [= 1/(dt*Iupc*IKC)]
         dtc -   double seconds   Iupc*Dtime
         IIMUF - integer unitless   flag to indicate if NFF uses
                 the IMU measurements (0:don't use, !=0:use)
         IRSDF - integer unitless   flag to indicate where the
                 input measurements are obtained from. (0:body mounted
                 accelerometers and rate gyros, !=0:RSDIMU)
         IRSDFY- integer unitless   flag to indicate if the RSDIMU
                 computed attitudes are to be used by the NFF (0:don:t
                 use, else if IRSDF!=0 & IIMUF=0 & IRSDFY!=0 : then use
                 them)

MXRPLF- integer unitless   the maximum number of sensor
        replications used in the NFF and in the FINDS FDI
        logic - currently limited to 2.

IREPLF- integer unitless   vector of sensor replications
        used by the NFF. The array index is by absolute
        sensor type, and the value is the replication count of
        that sensor used by the NFF

INOUTF- integer unitless   a matrix which indicates the
        status of all the sensors used by the NFF. The row
        index of INOUTF corresponds to the absolute sensor
        type, and the column index is the replication number
        of the sensor. The value of each element shows the
        current status of the sensor where:

                -3:   unavailable (selected out by decision
                      logic)
                -2:   failed
                -1:   available - but not used by the filter
                      (i.e. standby status)
                  0:   not available to the NFF
                  1:   available and used

refby: AVBIAS, BIASF, BLEND, CHKRAD, CLPSIO, DECIDE, DETECT, EKFN1,
        FILCOL, GTOI, INITF, INITG, MINSET, NAV, NOISR, PRNTIC,
        RECONF, RESCMP, SAVIT, SETISN, SUMIN, SUMOUT, UPDH

4.2.13   Description Of FLTCTL -
name: FLTCTL
cont: FINDS program control flags
vars: Ifilt - integer unitless   indicates form of NFF.
            Currently Ifilt=1, signifying only the standard EKF
            is used (i.e. not the square root form)

        IBfilt- integer unitless   indicates type of covariance
            propagation in the bias filter, where 1: propagate
            covariance, and 2: propagate information

        IgainP- integer unitless   frequency of Kalman filter
            gain printout (in samples/printout)

        Ipass - integer unitless   flag to control output to
            (binary) PLT file where 1: initialize PLT fie,

                    2: write data to file. 3: close PLT file
        Istop - integer unitless   not used currently
        Iwoc - integer unitless   indicates whether white or
              colored MLS noise corrections should be used in the
              NFF. (1: use colored noise assumptions [default].
              0: use white noise assumptions)
        Ierc - integer unitless   specifies whether or not
              corrections for earth's rotation are to be used bv NFF
              (1: use corrections [defalut]. 0: don't use
              corrections)
        Iysc - integer unitless   flag to indicate if
              measurements are to be scaled by Yscale. (-1: scale
              [default]. 0: don't scale)
        Hradar- double feet   altitude below which the radar
              altimeter is used bv the NFF in place of the MLS
              elevation sensor
refbv: BIASF. BLEND. CHKRAD. DETECT. EKFN1. FILCOL. FINDS. INITF.
        INITG. NAV. PRNTIC. SAVIT. SUMIN. SUMOUT. UPDB

4.2.14 Description Of FTITL1 -
name: FTITL1
cont: To store the comment records to be stored in the file header
        of the PLT file
vars: nline - integer unitless   number of 56 character comment
                records
        mxlin1- integer unitless   maximum number of columns
                in LTITL1
        LTITL1- integer string   comment records are stored in
                the columns of LTITL1. LTITL1 is dimensioned 15 bv
                mxlin1
refby: FILER1. FINDS. INITG. PRNTIC

4.2.15 Description Of GBLEND -
name: GBLEND
cont: No-fail filter blender gain (see [3] and [4])
vars: VBO -   double mixed   no-fail filter blender gain
refby: BIASF. BLEND, CLPSBE. INITG. NAV, RECONF

4.2.16 Description Of HEALCM -
name: HEALCM
cont: Quantities used by the healer logic (see section 2.5 in [2])
vars: kCThlr- integer unitless   contains a running count of
                the elapsed samples since the start of the current
                healer window
        kMXhlr- integer unitless   total number of samples to
                process before a healer window should be reset
        CONFBD- double unitless   logarithm of the initial
                confidence bound (1/19) for the healer test
        PhealT- double mixed   vector containing standard
                deviations of the expected noise (in program units)
                for each sensor type - to be used exclusively by the
                healers (this allows flexibility in specifying the
                sensor noise statistics appropriate to the healers -
                i.e. different from the detectors, no-fail filter, and
                simulation.) PhealT is ordered by absolute sensor
                type
        Bthrsh- double mixed   vector of largest expected
                normal operating biases for each sensor type (in
                program units). Bthrsh is only used in the healer
                logic and is ordered by absolute sensor index
        Fthrsh- double mixed   vector of smallest expected
                failure levels for each sensor type (in program
                units). Fthrsh is only used in the healer logic and
                is ordered by absolute sensor index
        Dthrsh- double mixed   vector of a decision
                thresholds to be applied to each healer process.
                This vector is ordered by sensor type.
                Dthrsh is defined as:
                    $Dthrsh(i) = 2.0*CONFBD*PhealT(i)**2$

143

refby: CHKRAD, DETECT, HEALR, INITG, PRNTIC, RECONF

4.2.17   Description Of HFCOM -
name: HFCOM
cont: Common quantities used by the healing and failure -
         reconfiguration logic in FINDS.
vars: Nfail - integer unitless   total number of sensors that
                 FINDS has determined to be "failed"
         NfailM- integer unitless   The maximum number of failures
                 that FINDS can process (i.e. dimension of IfailT &
                 IfailR)
         NNfail- integer unitless   number of new failures, i.e.
                 the incremental number of sensors which have just been
                 detected as failed - but have not been removed by the
                 reconfiguration logic
         Nheal - integer unitless   total number of sensors which
                 the healer logic has declared healthy at the end of a
                 healer window
         NhealM- integer unitless   the maximum number of sensors
                 which can heal in one instant. (i.e. the dimension
                 of IhealP)
         IfailT- integer unitless   vector containing the absolute
                 sensor type for each failed sensor. Whenever a sensor
                 fails its absolute sensor type (from Table 5) is added
                 to IfailT. Therefore, this vector is ordered by
                 relative time of occurrence of the failure. (failed
                 sensor index)
         IfailR- integer unitless   vector containing the
                 replication number for each failed sensor. It is
                 ordered the same as IfailT. Together IfailT(i) and
                 IfailR(i) determine the i'th failed sensor's type and
                 replication.
         IhealP- integer unitless   vector containing a list of
                 the failed sensors which have healed. The value of
                 an element is the index in IfailT and IfailR of the
                 healed sensor. Therefore, IhealP(j) represents the
                 j'th healed sensor and it (i.e. the value of

r

.

                    IhealP(i)) points to the IhealP(j)'th failed sensor
                    in IfailT and IfailR.
refby: CHKRAD. DECIDE. INITG. HEALR. RECONF

4.2.18   Description Of INITVL -
name: INITVL
cont: Initial values for the no-fail filter
vars: INOBPS- integer unitless   INOBPS=INOBP at the start of
                 the run
         PBFOI - double mixed   initial values for the
                 standard deviations of the bias free estimation error
                 (in user units). Addressed by absolute state index
                 (see Table 1)
         PBFIC - double mixed   initial values for the
                 standard deviation of the detector error information
                 (in user units), addressed by absolute sensor index
                 (see Table 5)
refby: CLPSBE. CLPSIO. DETECT. INITG. PRNTIC. RCOV. RECONF. SAVIT

4.2.19   Description Of INOU -
name: INOU
cont: Contains Fortran unit numbers for I/O to the users terminal
         and to all disk files.
vars: kin - integer unitless   unit no. for input from TTY
         kout - integer unitless   unit no. for output to TTY
         kdsk1 - integer unitless   unit no. for output to PLT
                 file
         kdsk5 - integer unitless   unit no. for input from
                 general input file
         kdsk6 - integer unitless   unit no. for output to OUT

                                   145

```
                   file
         kdsk7 - integer unitless      unit no. for input from filter
                   input file
         kdsk8 - integer unitless      unit no. for output to TLN
                   file
         kdsk11- integer unitless      unit no. for output to SUM
                   file
         kdsk12- integer unitless      unit no. for input from sensor
                   input file
refby:   AIRSPS, ATITGS, AUTLD, BANKTR, BIASF, BMLAS, BMRGS, CHKRAD,
         DECIDE, DGATIO, FILER1, FINDS, GETMLS, GMINV, ILNAC1,
         INITF, INITG, IRATG1, NAV, RADALS, RCOV, SAVIT, STABCN,
         TLOUT, VMPRT, WAYPNT
```

4.2.20   Description Of MAIN1 -
name: MAIN1
cont: Provides common dimensioning information for two dimensional
         arrays and a scratch area for temporary use by all
         subroutines.
vars: ndim - integer unitless   row dimension for two
                    dimensional arrays
         ndim1 - integer unitless $ndim + 1$
         COM1 - double   temporary   scratch array dimensioned
                   ndim by ndim
refby: ASUMER, BIASF, BLGAIN, DETECT, DGATIO, DOT2, DOT3, EQUATE,
         FILCOL, GETMLS, GMINV, GTOI, IMSCLE, IMTCG2, INITG, LKF,
         LRT, MADD1, MADDI, MAT1, MAT1A, MAT2, MAT3, MAT3A, MAT4,
         MAT5, MAT6, MATCG2, MATNUL, MMUL, MMUL2, MSCALE, NAV,
         OUTDAT, PRNTIC, RCOV, SAVIT, SEQNCE, STARTF, SUMIN, SUMMER,
         TRANSP, UPDA, UPDAB, UPDB, UPDH, UPDPH, UPDO, VADD1, VMAT1,
         VMAT2, VMPRT,

```

4.2.21   Description Of MAIN2 -
name:    MAIN2
cont:    Provides a temporary scratch array for use by all routines.
vars:    COM2 - double  temporary      scratch array dimensioned
                        ndim by ndim
refby:   BIASF. BLEND. BLGAIN. DETECT. DECIDE. EKFN1. INITG. NAV.
         PRNTIC. SAVIT

4.2.22   Description Of MULTDT -
name:    MULTDT
cont:    Quantities used in detecting multiple simultaneous failures
vars:    PRIORJ- double  mixed      vector of the logarithms of
                        the prior probability of more than one MLS sensor of
                        the same type to fail in the same instant (common mode
                        failure) (ordered MLS azimuth. elevation. range)
         ALamdJ- double  mixed      vector of the log-likelihood
                        of a multiple MLS sensor failure.  Ordered the same
                        as PRIORJ
         RESBJ - double  mixed      matrix of multiple MLS failure
                        compensated residuals vectors.  Columns are ordered
                        the same as elements of PRIORJ.
refbv:   DECIDE. DETECT. INITG. PRNTIC

4.2.23   Description Of SENSCM -
name:    SENSCM
cont:    Quantities used in determining the SIMULATED sensor
         configuration
vars:    IIMUS - integer unitless      flag to indicate if the IMU
                        sensor signals are simulated (available to the NFF)
                        where 1: IMU exists. and 0: IMU doesn't exist

147

IRSDS - integer unitless      flag to indicate if the RSDIMU
     is simulated. Where 1: simulated: and 0: not
     simulated
MXRPLS- integer unitless      the largest maximum number of
     replications of any sensor that was simulated
IREPLS- integer unitless      vector whose elements indicate
     the simulated replications of accelerometers, rate
     gyros, MLS, IAS, and IMU sensor systems, respectively
IREADS- integer unitless      not used currently

refby: CHKRAD. DETECT. INITG, FILCOL

4.2.24   Description Of SIMCOM -
name:     SIMCOM
cont:     Provides communication between the simulation and the routines
        used to record the PLT file (RECRDS and FILER1)
vars:     ifq -    integer        not used
        thalf - real         not used
        time -   real    seconds      current simulation time
        delt -   real    seconds      simulation integration step
                size
        nstep - integer        not used
        tstart- real     seconds      starting time of the
                simulation
        tstop - real     seconds      final time (estimated)
refby: CHKRAD. CKUNST. DECIDE. FILER1. FINDS. GYROCR. INITG.
        NAV. PRNTIC. RECONF. RECRDS, TLOUT

4.2.25   Description Of SMPRM -
name:     SMPRM
cont:     Saves general simulation quantities associated with the

PLT file
```
vars: nbuf - integer unitless   length of the physical record
                used to incrementally store the data in the PLT file
        ntick - integer unitless   the ratio of no. of simulation
                steps/record steps
        delt - real   seconds   the integration step size used
                by FINDS
        LCODE - integer   not used
refbv: READRC
```

4.2.26  Description Of STITL -
name:   STITL
cont:   Stores the comment records contained in the file header
        of the PLT file
```
vars: nline - integer unitless   number of 56 character
                comment records
        mxline- integer unitless   maximum number of columns
                in LTITL
        LTITLE- integer string   comment records are stored in
                the columns of LTITLE. LTITLE is dimensioned 15 bv
                mxline
refbv: READRC
```

4.2.27  Description Of SYSU1 -
name:   SYSU1
cont:   Quantities associated with the inputs to the no-fail filter.
```
vars: NU -   integer unitless   total number of inputs to
                no-fail filter including gravity inputs
        NU1 -   integer unitless   total number of inputs to
                no-fail filter associated with an input sensor (i.e.
```

NU-NG)
```
        NU1P1 - integer unitless   NU1+1
        NG -   integer unitless   total number of gravity inputs
        NU1C -   integer unitless   NU1 - currently not used
        INOUP - integer unitless   pointer vector to absolute
                 input measurements used by the NFF. Where, the array
                 index corresponds to the location in UF1, and the
                 value is the absolute input measurement type index
                 found in Table 3. Note: since we do not allow the
                 input vector to collapse this array is not strictly
                 required - however, it does provide much of the
                 functionality needed to facilitate reconfiguring the
                 inputs to the NFF in future releases of FINDS.
        UF1 -   double   mixed   vector of compensated inputs
                 used by the no-fail filter (computation in SUMIN)
refby: ADJTBP, AVECMP, BIASF, BLEND, BLGAIN, CLPSBE, CLPSIO, DECIDE,
        DETECT EKFN1, FILCOL, GTOI, HEALR, INITF, INITG, NAV, NOISR,
        PRNTIC, RCOV, RECONF, RESCMP, SAVIT, SETISN, SUMIN, UPDPH
```

4.2.28  Description Of SYSX1 -
name:   SYSX1
cont:   Bias free filter state dimensions and system matrices
vars:   NX -     integer unitless        total number of states in bias
                 free portion of the no-fai filter
        NX1 -    integer unitless        NX+1
        AF1 -    double   mixed          describe state transition
                 matrix - set in UPDA.  (see 2.2.13 in [2])
        BF1 -    double   mixed          discrete processes noise
                 covariance matrix (i.e. EWE'). (see 2.2.13 in [2])
refby:  ADJTBP, BIASF, BLEND, BLGAIN, CLPSBE, CLPSIO, DETECT, EKFN1,
        INITG, NAV, PRNTIC, RCOV, RECONF, SAVIT, STARTF

4.2.29   Description Of SYSXBO -
name:    SYSXBO
cont:    Quantities associated with the bias filter portion of the
         NFF.
vars:    NB  -    integer unitless        the current number of biases
                  estimated by the NFF (NB=NUB+NYB)
         NUB -    integer unitless        the current number of input
                  sensor biases estimated by the NFF
         NUB1 -   integer unitless        NUB+1
         NYB -    integer unitless        the current number of
                  measurement biases estimated by the NFF
         NBMXI -  integer unitless        the original (total) number
                  of biases requested to be estimated by the NFF
         INOBP -  integer unitless        pointer vector to the sensor
                  type of each bias estimated, where the array index is
                  the bias index used by the filter, and the value of
                  each element is the absolute sensor index (from
                  Table 5) of the corresponding sensor
         ABF1 -   double  mixed           discrete state transition
                  matrix which accounts for the estimation of normal
                  operating biases.  (see eq. 2.2.30 in [2])
refby:   ADJTBP, BIASF, BLEND, BLGAIN, CHKRAD, CLPSBE, CLPSIO, DETECT
         EKFN1, INITG, KALMN, NAV, PRNTIC, RCOV, RECONF, RESCMP, SAVIT,
         SUMIN, UPDPH

4.2.30   Description Of SYSYW1 -
name:    SYSYW1
cont:    Quantities associated with the no-fail filter's observations
         and process noises
vars:    NY  -    integer unitless        total number of possibly
                  averaged (or collapsed) measurements presented to the
                  no-fail filter
         Ndistb-  integer unitless        total number of process noise
                  inputs to the NFF
         NYMXI -  integer unitless        initial (maximum) number of
                  averaged measurements to the NFF
         INOYP -  integer unitless        pointer vector to "active"

                    averaged outputs used by the NFF where INOYP is formed
                    such that the array index corresponds one-to-one with
                    the elements of the (possibly collapsed) measurements
                    of the NFF, and the value of each element corresponds
                    to the absolute measurement index in Table 2.

INOYPI- integer unitless       inverse mapping of INOYP. i.e.
        the array index is the absolute measurement index, and
        the value is the corresponding index in the current
        measurement vector to the NFF.  If a particular
        measurement type is not used by the filter its value
        in INOYPI will be zero

YF1 -   double   mixed           vector of averaged
        measurements used by the NFF - uses absolute
        measurement sensor indexing

RF1 -   double   mixed           vector of measurement noise
        covariances organized by absolute measurement index
        (Table 2).  Each element in RF1 is adjusted to
        reflect the number of sensors averaged

OF1 -   double   mixed           vector of process noise
        covariances organized by absolute input index
        (Table 3). (see 2.2.14 in [2])

HP1 -   double   mixed           effective observation matrix
        for NFF (partial of h w.r.t.x) (see 2.2.31 in [2])

refby: ADJTBP, BIASF. BLEND. BLGAIN, CHKRAD. CLPSBE. CLPSIO. DETECT.
       EKFN1, INITF, INITG, NAV, NOISR, RCOV. SVIT. SUMOUT, UPDPH


4.2.31  Description Of YOBSRV -
name:    YOBSRV
cont:    Contains the scaling array for the filter observations
vars:    Yscale- double   mixed         vector of scale factors used
             to scale each averaged measurement into the NFF. The
             scaling is performed to ensure that the measurement
             noise variance is unity for each sensor
refby: AVECMP. BIASF. BLGAIN. CLPSBE. DETECT. INITF, INITG, RCOV
       RESCMP, SAVIT, SUMOUT, UPDH, UPDPH

5   REFERENCES

[1]   Lancraft. R.E. and Caglavan. A.K.. "FINDS: A Fault
      Inferring Nonlinear Detection Svstem - User's Guide".
      NASA CR-172199. September 1983.

[2]   Caglavan, A.K. and Lancraft. R.E.. "A Fault Tolerant Svstem
      for an Integrated Avionics Sensor Configuration". NASA CR-3834.
      1984.

[3]   Caglavan, A.K. and Lancraft. R.E.. "An Aircraft Sensor
      Fault Tolerant System". NASA CR-165876. April 1982.

[4]   Caglavan. A.K. and Lancraft, R.E.. "A Separated Bias
      Identification and State Estimation Algorithm for
      Non-Linear Svstems". Automatica, Vol. 19, No. 5.
      pp. 561-570. September 1983.

[5]   "VAX-11 DIGITAL Standard Runoff Version 2.0. Users Guide".
      Digital Equipment Corporation. No. AA-J268B. May 1982.

APPENDIX A

SUMMARY OF SPECIFIC HARDWARE AND SOFTWARE REQUIREMENTS


Computer:       Digital Equipment - VAX-780 or 750
Storage:        At least one disk drive
Terminals:      Either a Tektronix model 4010/4014 or one that emulates a
                Tektronix 4010/4014 (for plotting purposes.)
Hard-copy
devices:        No specific requirements.  All output is directed to
                disk files or to the users terminal.
Operating
System:         VMS Version 3.0 or higher.  VMS utilities and
                libraries are required.
Software:       Fortran-77 compiler

The following files are supplied:

        Command Files:
                FINDS.COM
                FINDSLIB.COM
                GETDOC.COM
                MAKEFPG.COM

        Executable Files:
                DOC.EXE
                FINDS.EXE
                PLOTD.EXE
                PRINTD.EXE

        Fortran Files:
                DOC.FOR
                FGAC.FOR
                FIO.FOR

# SUMMARY OF SPECIFIC HARDWARE AND SOFTWARE REQUIREMENTS

```
FIOSUB.FOR
FMAIN.FOR
FSENS.FOR
FSFDI.FOR
FUTSUB.FOR
FVMSUB.FOR
FWIND.FOR
PLOTD.FOR
PRINTD.FOR
```

Libraries:
```
FINDSLIB.OLB
```

OPT Files:
```
PLOT.OPT
PRINTD.OPT
```

COM files:
```
FINDSC.COM
FINDSL.COM
FINDSLIB.COM
FOREIGN.COM
FPMV3.COM
```
FIL files:
```
FINDSPM.FIL
FINDSPMA.FIL
FINDSPMB.FIL
```
RNO files:
```
FPMV3.RNO
```
LISA floppy disk files:
A 5 1/4 floppy disk which contains two Lisa Draw applications:
1. FINDSfigs - all the flowcharts in the FINDS programmers
   manual.
2. FINDScharts - all the tables used in the FINDS programmers
   manual.

APPENDIX B

GENERATING THE FINDS PROGRAMMERS MANUAL

It was stated in the introduction that this manual was generated in a semi-automatic fashion using a combination of a rudimentary text formatting program called Digital Standard Runoff (DSR), an Apple Lisa computer using LisaDraw, and a text stripping program (DOC). This appendix provides the details of this system. In particular this appendix will:

   o Enumerate the procedure required to produce a copy of this programmers manual.

   o Briefly describe the mechanics of the automatic documentation system.

   o Document the current set of "rules" for embedded source code documentation.

   o Describe the steps required for adding future documentation to the manual.

   o And finally, present some observed stengths and shortcomings of this approach.

The following steps are required to produce a copy of this manual:

   1. Type $ @foreign and then $ @findspg , this produces a file fpmv3.mem which can be printed on a suitable printing device (daisy wheel, dot matrix, or laser printer).

GENERATING THE FINDS PROGRAMMERS MANUAL

2. Load the Lisa floppy disk into an Apple Lisa computer and
    print all the figures and tables using LisaDraw, or
    alternatively if a file of current figures is maintained,
    simply make copies of the figures and tables.

3. Insert the figures and tables into the document in the
    approriate places.

As shown above, only three simple steps are required to produce a copy
of an existing manual. Now let's take a closer look at what actually
was performed in step 1. Figure 13 shows a closeup of the underlying
mechanics. From this figure we can see that step 1 first stripped,
from a list of files, documentation containing:

o Each file's contents

o Subroutine descriptions

o Common block descriptions

In addition, index items were added and the files were put in a form
compatible with DSR (all done via the program DOC.) Each type of
information is saved in a separate file. These files are then
referenced in a runoff file which contains a template of the manual
(e.g. the Introduction, Appendicies, and beginnings of chapters where
the files will be included.) The output of DSR is a file which
contains all the written text, and saves "white" space for all the
figures and tables.

Figure 13. Mechanics of Automatic Manual Generation

GENERATING THE FINDS PROGRAMMERS MANUAL

In order for the text stripping program, DOC, to work properly various
"rules" are required for placing embedded documentation in Fortran
source files. The basic premise is that comment lines which occur
between special header lines are to be treated as documentation.
Currently the following header delimiters are supported:

    1. Cfil ... Cendfil - These bracket file content comments.

    2. Cdoc ... Cenddoc - These bracket subroutine documentation.

    3. Ccom ... Cendcom - These bracket common block documentation.

In general, formatting within header delimiters is arbitrary.
However, if "Cname:" is encountered, the rest of the line is treated
as a file, subroutine or common block name and is entered into the
index. Furthermore, if figures are required, DOC can be used to save
space for them. This is done by using the following construct:
"Cfig: FFPF(n)" where $1 < n < 99$ is the number of Floating Full Page
Figures required. The following rules must be followed:

    1. Header delimiters must occur on separate lines

    2. Each line between header delimiters must start with a comment
       character "C" (for FORTRAN files).

    3. Each line of documentation must be less than or equal to 70
       characters (not counting the comment character.)

    4. Each line is passed verbatum into the document so that block
       formatting can be performed using tabs and spaces.

    5. The special symbols "Cname" and Cfig" discribed above must
       start in column 1.

Although the formatting of documentation is arbitrary and therefore up
to the programmer, in order for the final document to be consistent a
documenting convention is required. In this document the following
convention was adhered to:

File header documentation

```
Cfil
Cname:  Name of the file.
Ccont:  Description of what the file contains.
Csubr:  List of subroutines with a one line description of each.
Cendfil
```

Subroutine and Program documentation

```
Cdoc
Cname:  Name of subroutine or function (with optional enumeration of
        Mnemonics.)
Cfunc:  Short description of how the subroutine functions.
Ccall:  Sample calling sequence.
Cargs:  List of arguments used in calling sequence along with a short
        definition and an indication of whether they are used as inputs
        outputs or both.
Cints:  List of key internal variables and there definitions.
Crefs:  List of subroutines referenced by the routine (with optional
        description of each.)
Crefby: List of the routines that reference this routine.
Ccomm:  List of the common blocks used by this routine (with optional
        description of each.)
Cfig:   FFPF(n) TITLE[...]
Cenddoc
```

Common block documentation

```
Ccomblk
Cname:  Name of the common block.
Ccont:  Purpose or contents of the common block.
Cvars:  List of variables along with a definition of each.
Crefby: List of routines containing this common block
Cendcom
```

GENERATING THE FINDS PROGRAMMERS MANUAL

At this point the reader should have a basic understanding of how the automatic documentation system works, as well as how to add documentation to a Fortran source file (the reader is refered to [5] for a complete discussion of the DSR commands used to format the runoff file.) However, it is not yet clear what steps are required to generate the manual if additional documentation, and/or figures or tables are added or deleted. Basically the same steps mentioned at the start of this appendix apply, with the following exception: since DSR doesn't support the notion of lists of tables, figures, or references -- they must be maintained by hand. This means that:

1. If table or figure numbering is modified, all direct references to figures or tables by number must be updated to reflect the new sequence. This can be avoided if they are refered to by title or placement.

2. The page numbers in the list of figures and the list of tables must be corrected. This is accomplished by simply running DSR once, noting the correct page numbers, correcting the runoff file, and re-running DSR.

3. If references are added or subtracted from the manual, citations made in the body of the report must be corrected, as well as the list of references itself.

In closing, the following observations are made as to the effectiveness of this method. On the positive side:

1. The documentation is available inside the source file itself, therefore it is readily available for a programmer to reference.

2. Documentation can be kept up to date by simply making an incremental addition to existing text. Moreover it can conveniently be done at the same time the code is altered - when the concept is clearest.

3. Working documentation can be made available at any point in the development process.

4. It's flexible. All documentation is stored electronically. Therefore, changes can be made without disturbing the overall structure, or compleatly re-drawing old figures to add minor

modifications.

For all its strong points, there are of course some weaknesses as
well:

1. DSR's lack of support for tables, figures, and references can
    create some extra maintainence effort. This could be
    eliminated if a more powerful text formatter where used, or
    if a pre-filter were written to do the maintainence
    automatically.

2. Because the documentation is written in a decentralized
    fashion it can be discontinuous in style and notation, unless
    clearly defined rules are followed.

3. If a correction in notation is desired several files must be
    modified, rather than a single one in the case of a more
    conventional document.

4. Source files will, of course, be larger if this method is
    used. This may be a concern if disk space is at a premium.

INDEX

1. Report No. NASA CR-177986	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle FINDS: A Fault Inferring Nonlinear Detection System Programmers Manual Version 3.0		5. Report Date December 1985
		6. Performing Organization Code
7. Author(s) Roy E. Lancraft		8. Performing Organization Report No. Report No. 6012
9. Performing Organization Name and Address BBN Laboratories 10 Moulton Street Cambridge, MA 02238		10. Work Unit No.
		11. Contract or Grant No. NAS1-16579
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington DC 20546		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code 505-34-13-12

15. Supplementary Notes

Langley Technical Monitor: Frederick Morrell

Computer Program Documentation

16. Abstract

This report provides detailed software documentation of the digital computer
program FINDS (Fault Inferring Nonlinear Detection System) Version 3.0. FINDS
is a highly modular and extensible computer program designed to monitor and
detect sensor failures, while at the same time providing reliable state estimates.
In this version of the program the FINDS methodology is used to detect, isolate,
and compensate for failures in simulated avionics sensors used by the Advanced
Transport Operating Systems (ATOPS) Transport System Research Vehicle (TSRV) in
a Microwave Landing System (MLS) environment. It is intended that this report
serve as a programmers guide to aid in the maintanence, modification, and
revision of the FINDS software.

17. Key Words (Suggested by Authors(s)) Sensor failure detection, fault tolerant navigation, MLS, no-fail filter	18. Distribution Statement Unclassified – Unlimited Subject Category 06		
19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21 No. of Pages 173	22. Price A08

End of Document