DAO Office Note 1998-05 Version 1 Dated 12/06/1999

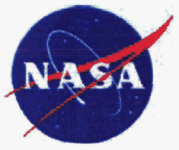
Office Note Series on Global Modeling and Data Assimilation

Siegfried Schubert, Editor

Documentation of the Physical-Space Statistical Analysis System (PSAS) Part III: The Software Implementation

J. W. Larson*
J. Guo†
G. Gaspari†
A. da Silva
P. M. Lyster*

Data Assimilation Office, Goddard Laboratory for Atmospheres
† *General Sciences Corporation, Laurel, Maryland*
* *Earth System Science Interdisciplinary Center (ESSIC), University of Maryland*



Robert M. Atlas, Acting Head
Data Assimilation Office
Goddard Space Flight Center
Greenbelt, Maryland 20771

Revision History

This Office-Note 98-05 was first produced for the PSAS Software Workshop on October 26, 1998.

Version Number	**Version Date**	**Pages Affected/ Extent of Changes**	**Aproval Authority**
Version 1β	October 26, 1998	First draft (before review) from PSAS Workshop	
Version 1	December 6, 1999	Initial release	ON Editor

Abstract

We describe the software implementation of the Physical-space Statistical Analysis System (PSAS) version v1.5.1 at the NASA Data Assimilation Office (DAO). This software implements a statistical algorithm that combines irregularly spaced observations with a gridded forecast to produce an optimal estimate of the state of the atmosphere. We describe how the observational data, and their attributes, as well as the error covariance matrices are managed during the life cycle of the algorithm: this is the main source of *software complexity* of the PSAS. This is mainly due to the diversity of data types and sources, as well as the use of the multivariate formulation.

An on-line version of this document can be obtained from

ftp://dao.gsfc.nasa.gov/pub/office_notes/on199805v1.1.ps.Z (postscript)

Visit also the Data Assimilation Office's Home Page at

http://dao.gsfc.nasa.gov/

Contents

Revision History ii

Abstract iii

List of Figures viii

List of Tables x

1 Preface 1

2 Introduction 2

3 Basic Aspects of the PSAS 5
3.1 Coordinate Systems Used in the PSAS 5
3.1.1 Geographic Spherical Coordinates 5
3.1.2 Observation Attributes 5
3.1.3 The Regular Latitude/Longitude Grid 7
3.1.4 The Quasi-Equal Area Grid 8
3.2 Block Decomposition of Matrices 10
3.3 Operator Formulation of PSAS 11
3.4 Data Management in PSAS 17
3.4.1 PSAS Data Flow 17
3.4.2 The Covariance Data Life Cycle 17
3.4.3 The PSAS Resource File 20

4 Top Level Control Flow — `getAIall()` 21
4.1 Analysis Interfaces to the PSAS 21
4.2 Description of Variables and Include Files used in `getAIall()` 21
4.3 Control Flow through `getAIall()` 23
4.3.1 Stage I Processing of Covariance Data 23
4.3.2 Processing of Observational Data 26
4.3.3 Stage II Processing of Covariance Data 27
4.3.4 Stage III Processing of Covariance Data–Calculation of Analysis Increments 29

5 Stage I: Initialization and Processing of Covariance Data 31
5.1 Overview of the Initialization Routine `initRSRC()` 31
5.2 Observation Type and Source Attribute Tables—`ktname0()` 33
5.3 Observation Data Source Tables—`kxname0()` 33
5.4 Determination of Observation Error Classes and State I data for σ_{ou} and σ_{oc}—`set_OEclas()` 35
5.5 Initialization of State I Observation Error Vertical Correlation Tables . . . 40
5.5.1 State I Observation Error Vertical Correlations–`set_OEvCor()` . . . 40

5.5.2 State I Observation Error Horizontal Correlations–`set_OEhCor()` . . 42
5.6 State I Forecast Error Correlation Data 46
5.6.1 State I Forecast Error Vertical Correlations—`set_FEvCor()` 46
5.6.2 State I Forecast Error Horizontal Correlations—`set_FEhCor()` . . . 47
5.7 State I Data for σ^{ψ} and σ^{χ}—`tabl_FEsigW()` 51
5.8 State I Data for the Geostrophic Balance Parameters α_{um}, α_{ul}, α_{vm}, and α_{vl}—`tabl_FEalpha()` 52

6 Processing of Observation Attributes **55**
6.1 Selection of Observations for Analysis—`restrict()` 55
6.2 Regional Domain Decomposition and Sorting of and Observation Attributes—`sort()` 56
6.3 Elimination of Duplicate Observations—`dupelim()` 58
6.4 Superobbing—`proxel()` 62
6.5 Separation of Eliminated and Retained Observations—`tofront()` 64

7 Stage II: Processing of Covariance Data **68**
7.1 Interpolation of Observation Error Standard Deviations from State I Tables — `intp_sigO()` 68
7.2 Determination of the Characteristics of the IMAT Tables 70
7.2.1 Vertical Level Entries — `merg_plevs()` 70
7.2.2 Latitudinal Entries — `merg_lats()` 72
7.3 Calculation of Forecast Error Correlation IMAT Tables 74
7.3.1 State II Forecast Height and Wind Error Correlation Tables — `set_fecHH()` 74
7.3.2 State II Forecast Water Vapor Mixing Ratio Error Correlation Tables — `set_fecQQ()` 77
7.4 Observation Error Univariate Correlation Tables — `set_oecHH()` 81
7.5 Calculation of IMAT Tables for α_{um}, α_{ul}, α_{vm}, and α_{vl} — `imat_alpha()` . 83
7.6 IMAT Tables for σ^{ψ} and σ^{χ} — `imat_sigW()` 84
7.7 Forecast Error Standard Deviations 85
7.7.1 Input of Gridded Forecast Error Standard Deviation Data — `getsigF()` 85
7.7.2 Gridded Fields of the Elements of Σ^h — `dervsigF_slD()` 87
7.7.3 Gridded Fields of the Elements of Σ^h — `dervsigF_upD()` 89
7.7.4 Interpolation of Elements of Σ^h — `intp_sigF()` 90

8 Stage III: Solution of the Innovation Equation — `solve4x()` **94**
8.1 Control Flow through `solve4x()` 94
8.2 The Conjugate Gradient Solver —`conjgr()` 95
8.3 Software Implementation of $(HP^fH^T + R)\mathbf{x}$ —`op_Mx()` 97

9 Stage III: Solution of the Analysis Equation **104**
9.1 Calculation of Analysis Increments — `getAinc()` 104
9.2 Software Implementation of P^fH^T — `mvPHx()` 108
9.2.1 V-vector Calculation of $P^fH^T\mathbf{x}$ — `op_Pf()` 114

10 Stage III: Software Implementation the "Lower-Level" Operators in PSAS120
10.1 Mass-coupled Height/Wind Forecast Error Covariances Γ^h 120
10.2 Mass-decoupled Height/Wind Error Covariances Γ^ψ, and Γ^χ 122
10.3 Forecast/Observation Error Standard Deviation Operator Σ 125
10.4 The Symmetric Error Correlation Operator – `sym_Cxpy()` 125
10.4.1 The Diagonal Block Operator `Cprod1()` 131
10.4.2 The Off-diagonal Block Operator `Cprodx()` 134
10.5 The Rectangular Error Correlation Operator C_{rec} 135
10.5.1 The Block Operator `gCprodx()` 139

Appendix A: The Analysis Interfaces `getAIall()` 143

Appendix B: Complete Procedural Flow Diagram 149

Appendix C: A Sample Resource File `psas.rc` 161

Appendix D: Complete Covariance Data Life Cycle Diagrams 181

Acknowledgments 188

References 189

List of Figures

1 The Control Diagram, or Calling Tree, of the PSAS. 4
2 The spherical coordinate system used in the PSAS. 6
3 Typical geographic distribution of observations over a six-hour interval that are input into the PSAS. 8
4 Icosahedral decomposition used in the PSAS, with region numbering. 9
5 Regional decomposition and sorting hierarchy that is applied to each observation attribute vector. 10
6 QEA horizontal grid longitude vector and indexing arrays. 11
7 Block decomposition of matrices in the PSAS. 12
8 Data life cycle for parameters used in PSAS. 19
9 Top-level control flow below analysis interfaces `getAIall()`. 25
10 Calling tree for the initialization routine `initRSRC()`. 32
11 Calling tree for `ktname0()`. 34
12 Calling tree for `kxname0()`. 35
13 Calling tree for `setOEclas()`. 39
14 Observation error standard deviation data structures. 39
15 Calling tree for `set_OEvCor()`. 42
16 Data structures containing State I tables for ν_{ou} and ν_{oc}. 43
17 Calling tree for `set_OEhCor()`. 44
18 Data structures containing State I Data for ρ_{oc}. 45
19 Calling tree for `setFEvCor()`. 47
20 State I data for upper-air height/wind (sea-level pressure/wind) forecast error vertical correlations. 48
21 State I data for water vapor mixing ratio forecast error vertical correlations. 48
22 Calling tree for `setFEhCor()`. 50
23 State I data for upper-air height/wind (sea-level pressure/wind) forecast error horizontal correlations. 50
24 State I data structures mixing ratio forecast error horizontal correlations. . 51
25 Calling tree for `tabl_FEsigW()`. 52
26 State I data for the streamfunction and velocity potential forecast error standard deviations σ^{ψ} and σ^{χ}. 53
27 Calling tree for `tabl_FEalpha()`. 54
28 State I data for the geostrophic balance parameters α_{um}, α_{ul}, α_{vm}, and α_{vl}. 54
29 Calling tree for `restrict()`. 57
30 Calling tree for `sort()`. 59
31 Calling tree for `dupelim()`. 61
32 Calling tree for `proxel()`. 65
33 Calling tree for `intp_sig0()`. 69
34 Calling tree for `merg_plevs()`. 72
35 Calling tree for `merg_lats()`. 73
36 Calling tree for `set_fecHH()`, `set_fecQQ()`, and `set_oecHH()`. 79
37 Calling tree for `imat_alpha()`. 84
38 Calling tree for `imat_sigW()`. 85
39 Calling tree for `getsigF()`. 86
40 Calling tree for `dervsigF_slD()`. 88
41 Calling tree for `dervsigF_upD()`. 90

42 Calling tree for `intp_sigF()`. 93
43 Calling tree for the routine `solve4x()`. 96
44 Levels of the nested preconditioner. 96
45 Calling tree for `op_Mx()`. 103
46 Calling tree for `getAinc()`. 105
47 Organization of the QEA "G-vector" grid 108
48 Organization of the "V-vector" grid 114
49 Calling tree for `op_Pf()`. 118
50 Band structure of matrix operators in PSAS. 127
51 Control flow for `sym_Cxpy()`. 128
52 Calling tree for the operator `rec_Cxpy()`. 137
53 Data life cycle for the operator Γ^h. 182
54 Data life cycle for the operator Σ^h. 183
55 Data life cycle for the operators Σ^ψ and Σ^χ. 184
56 Data life cycle for the operators C^h, C^ψ, and C^χ. 185
57 Data life cycle for the operators Σ^o_u and Σ^o_c. 186
58 Data life cycle for the operators C^o_u and C^o_c. 187

List of Tables

1 Data type index **kt** values 13
2 Nonzero elements of the multivariate operator Γ^h 14
3 Nonzero elements of the multivariate operator Γ^ψ 14
4 Nonzero elements of the multivariate operator Γ^χ 14
5 Elements of the diagonal matrix Σ^h $(\sigma^{p_{sl}} = g\bar{\rho}\sigma^h)$ 15
6 Elements of the diagonal matrix Σ^ψ 15
7 Elements of the diagonal matrix Σ^χ 15
8 Nonzero elements of the operator C^h 16
9 Nonzero elements of the operator C^ψ 16
10 Nonzero elements of the operator C^χ 17
2 Data passed into **getAIall()** via its interface. 24
3 Data box attributes **boxes** returned by **setbox()**. 56
4 Data passed into **restrict()** via its interface. 57
5 Data passed into **sort()** via its interface. 59
6 Data passed into **dupelim()** via its interface. 61
7 Black list data structures from **proxel.h** 64
8 Data passed into **proxel()** via its interface. 65
9 Data passed into **tofront()** via its interface. 67
10 Data passed into **intp_sig0()** via its interface. 70
11 Data passed into **merg_plevs()** via its interface. 72
12 Data passed into **merg_lats()** via its interface. 74
13 Forecast Height/Wind Error Horizontal Correlation IMATs 77
14 Forecast Height/Wind Error Vertical Correlation IMATs 78
15 Forecast Height/Wind Error Correlation Normalization Factors 79
16 Objects from the module **FEalpha_imat** 84
17 IMATs for σ^ψ and σ^χ 85
18 Data passed into **getsigF()** via its interface. 87
19 Data passed into **dervsigF_slD()** via its interface. 88
20 Data passed into **dervsigF_upD()** via its interface. 90
21 Data passed into **intp_sigF()** via its interface. 93
22 Summary of arguments to **solve4x()**. 97
23 Summary of arguments to **op_Mx()**. 102
24 Arguments to the routine **getAinc()**. 109
25 Arguments to the routine **mvPHx()**. 113
26 Summary of arguments to **op_Pf()**. 119
27 Summary of arguments to **mv_Alf()**. 122
28 Summary of arguments to **mv_diag()**. 126
29 Summary of arguments to **sym_Cxpy()**. 132
30 Error Correlation Calculation Calls from **Cprod1()**. 132
31 Summary of arguments to **Cprod1()**. 133
32 Error Correlation Calculation Calls from **Cprodx()**. 135
33 Summary of arguments to **Cprodx()**. 136
34 Summary of arguments to **rec_Cxpy()**. The 'columns' arguments **n_krj**, ..., **qlj_y** are not shown for brevity. 140
35 Summary of arguments to **gCprodx()**. 142
36 Data passed into **getAIpuv()** via its interface. 144

37 Data passed into **getAIzuv()** via its interface. 146
38 Data passed into **getAImix()** via its interface. 148

1 Preface

This article describes the software implementation of the Physical-space Statistical Analysis System (PSAS) version v1.5.1. The software is used in the production GEOS DAS algorithm at the DAO. The articles describing the PSAS are as follows:

1. da Silva, A., and J. Guo, 1996: Documentation of the Physical-Space Statistical Analysis System (PSAS) Part I: The Conjugate Gradient Solver Version, PSAS-1.00. *DAO Office Note 96-02.*

2. Guo J., J. W. Larson, G. Gaspari, A. da Silva, and P. M. Lyster, 1998: Documentation of the Physical-Space Statistical Analysis System (PSAS) Part II: The Factored-Operator Formulation of Error Covariances. *DAO Office Note 98-04.*

3. Larson J. W., J. Guo, G. Gaspari, A. da Silva, and P. M. Lyster, 1998: Documentation of the Physical-Space Statistical Analysis System (PSAS) Part III: The Software Implementation. *DAO Office Note 98-05.*

4. da Silva A., M. Tippett, and J. Guo, 1999: The PSAS User's Manual. To be published as *DAO Office Note 99-XX.*

Future documents will discuss versions of PSAS under development. These will include: forward observation operators, advanced covariance models, advanced Fortran 90 features, and the ability to run on parallel distributed-memory computers using the Message Passing Interface (MPI).

2 Introduction

We describe the *software implementation* of the Physical-space Statistical Analysis System (PSAS) version v1.5.1 [Cohn et al., 1998, Staff, 1996]. The PSAS combines the forecast $\mathbf{w^f} \in I\!R^n$ and observations $\mathbf{w^o} \in I\!R^p$ to produce an optimal estimate of the true state state of the atmosphere. The PSAS performs this analysis using the following equations:

The innovation equation:

$$(HP^fH^T + R)\mathbf{x} = \mathbf{w^o} - H\mathbf{w^f}; \tag{1}$$

and

The analysis equation:

$$\mathbf{w^a} - \mathbf{w^f} = P^fH^T\mathbf{x}, \tag{2}$$

where P^f is an $n \times n$ forecast error covariance matrix, R is a $p \times p$ observation error covariance matrix, H is a $p \times n$ matrix that maps gridded forecast vectors to observations on an unstructured grid, $\mathbf{w^o}$ is a $p-$vector of observations, $\mathbf{w^f}$ is an $n-$vector of the gridded forecast, and $\mathbf{w^a}$ is an analysis $n-$vector. The right hand side of Eqn. (1) is called the innovation vector or the *observed-minus-forecast* (OMF) residual, and the left hand side of Eqn. (2) is called the *analysis increment* (AI). Equation (1) is solved using a pre-conditioned conjugate gradient (CG) algorithm [Golub and van Loan, 1989, da Silva and Guo, 1996]. With the current system ($n = 10^6$ and $p = 10^5$ are the approximate values) setting up and solving Eqn. (1) costs about half the computational effort in the PSAS. The remaining effort is taken by the transformation in Eqn. (2) of $\mathbf{x}$ from observation to state space [Cohn et al., 1998]. A discussion of the *computational complexity* of the PSAS and the end-to-end Goddard Earth Observing System Data Assimilation System (GEOS DAS) is given in [Lyster, 1998].

This article describes the control flow and data flow of the software implementation of the PSAS version v1.5.1. The PSAS uses a factored-operator formulation for the error covariance matrices as described in the companion document [Guo et al., 1998]. This formulation determines how the observational data, and their attributes, as well as the error covariance matrices are managed during the life cycle of the algorithm: this is the main source of *software complexity* of the PSAS. This is mainly due to the diversity of data types and sources, as well as the use of the multivariate formulation.

The PSAS performs the following **functions**:

1. Construction of the matrix $HP^fH^T + R$ and solution of Eqn. (1) for $\mathbf{x}$ by a preconditioned conjugate gradient method.
2. Construction of the matrix P^fH^T and calculation of the AI ($\mathbf{w^a} - \mathbf{w^f}$) from $\mathbf{x}$ using the mapping defined in Eqn. (2).
3. Calculation of forecast and observation error covariance data for use in functions 1 and 2.
4. Partitioning, sorting, and other processing of input observational data.

The functions labeled 1 and 2 are handled together by the *analysis interface routine* `getAIall()`. This subroutine is modified at compile time to generate three subroutines which are called by the application driver for the PSAS and which handle the three components of the analysis: the sea-level pressure-wind analysis `getAIpuv()`, the upper-air height-wind analysis `getAIzuv()`, and the water vapor mixing ratio analysis `getAImix()`.

Function 3 above involves the *covariance data life cycle*, a concept introduced in Section 3.4.2, and developed throughout the remainder of this document. Function 4 is needed in support of the higher level functions 1 to 3.

In Section 3 we introduce the basic concepts behind the PSAS software, including coordinate systems and the covariance data life cycle.

In Section 4 we present a discussion of the data and control flow for the analysis interface `getAIall()`.

In Sections 5 and 7 we describe the initialization and tabulation stages of covariance data life cycle.

In Section 6 we describe the observational data processing steps described in Function 4 above.

In Section 8 we describe the software implementation of the preconditioned CG solution of Eqn. (1), including the control software used to form HP^fH^T and R.

In Section 9 we describe the software implementation of the analysis equation Eqn. (2), including the control software used to form P^fH^T.

In Section 10 we describe the lower-level routines called to calculate HP^fH^T, R, and P^fH^T.

How to read this document:
The document [Guo et al., 1998] provides the theoretical basis for this software description and should be read first. The software of the PSAS is best understood through the *control flow* (or calling tree) and the *data flow* – the PSAS is partly asynchronous (e.g., data can arrive in any order) so a thorough understanding of both aspects of the algorithm is important. The control flow is relatively "easy" to understand and a course-grained representation is shown in Figure 1 – it may be used throughout this paper as a reference.

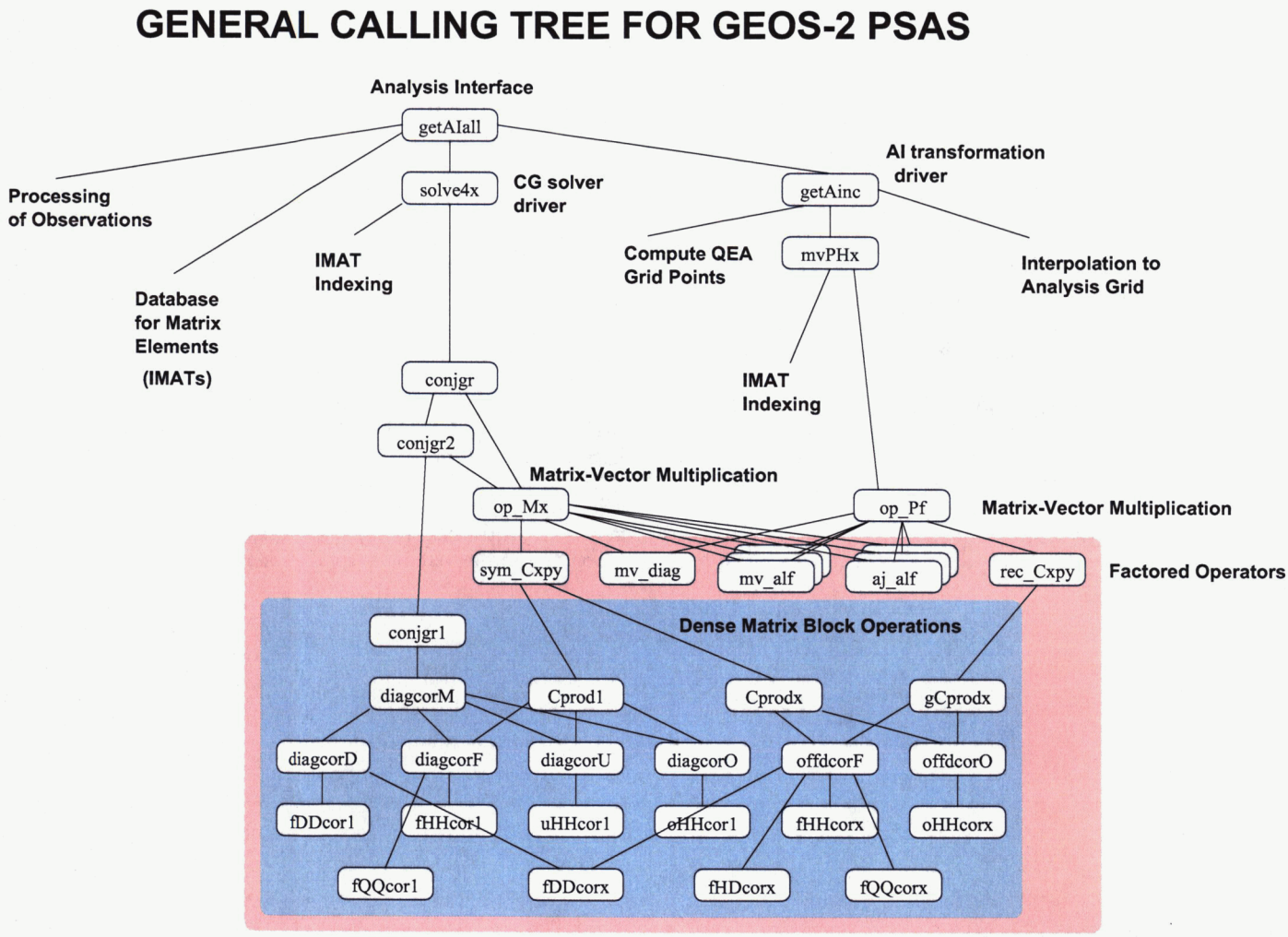


Figure 1: The Control Diagram, or Calling Tree, of the PSAS.

3 Basic Aspects of the PSAS

3.1 Coordinate Systems Used in the PSAS

3.1.1 Geographic Spherical Coordinates

In the PSAS v1.5.1, locations of forecast and observational data are defined in terms of geographic spherical coordinates latitude φ and longitude λ, with a pressure vertical coordinate p. The radial coordinate r is fixed, with $r = a = 6.376 \times 10^6$m. The unit vectors in the directions r, λ, and φ are $\hat{\mathbf{e}}_r$, $\hat{\mathbf{e}}_l$, and $\hat{\mathbf{e}}_m$, respectively. The *longitudinal* unit vector $\hat{\mathbf{e}}_l$ points to the East, while the *meridional* unit vector $\hat{\mathbf{e}}_m$ points to geographic North, as illustrated in Figure 2. The set of unit vectors $(\hat{\mathbf{e}}_r, \hat{\mathbf{e}}_l, \hat{\mathbf{e}}_m)$ form a right-handed triple, since

$$\begin{aligned} \hat{\mathbf{e}}_r \times \hat{\mathbf{e}}_l &= \hat{\mathbf{e}}_m \\ \hat{\mathbf{e}}_l \times \hat{\mathbf{e}}_m &= \hat{\mathbf{e}}_r \\ \hat{\mathbf{e}}_m \times \hat{\mathbf{e}}_r &= \hat{\mathbf{e}}_l. \end{aligned}$$

The horizontal error correlations between observation or forecast values at locations (φ_i, λ_i) and (φ_j, λ_j) are computed using the *polarity index* τ_{ij} which is defined as

$$\tau_{ij} = \hat{\mathbf{e}}_{r_i} \cdot \hat{\mathbf{e}}_{\mathbf{r_j}}. \tag{3}$$

More details of the coordinate system of the PSAS are presented in Section 3.2 of [Guo et al., 1998].

3.1.2 Observation Attributes

Observational data are input to the PSAS in the form of the *observed minus forecast* (OMF) vector $\mathbf{w^o} - H\mathbf{w^f} \in I\!R^p$. This input OMF vector and the following data form the set of *input observational attributes*:

- Variable type index `kt` (`INTEGER`), one of the set {1,2,...,`ktmax`}, where the value of `ktmax` is defined in the include file `ktmax.h`.
- Instrument index `kx` (`INTEGER`), one of the set {1,2,...,`kxmax`}, where the value of `kxmax` is defined in the include file `kxmax.h`.
- Region index `kr` (`INTEGER`, assigned at run-time), one of the set {1,2,...,`maxreg`}, where the value of `maxreg` (= 80 for PSAS version v1.5.1) is defined in the include file `maxreg.h`.
- Latitude φ (`REAL`), in degrees (−90,90).
- Longitude λ (`REAL`), in degrees (−180,180).
- vertical pressure level (hPa) (`REAL`).
- The OMF value $\mathbf{w^o} - H\mathbf{w^f}$ (`REAL`).

The PSAS returns the following *output observational attributes*:

- Forecast error standard deviations at observation locations σ_f (`REAL`).

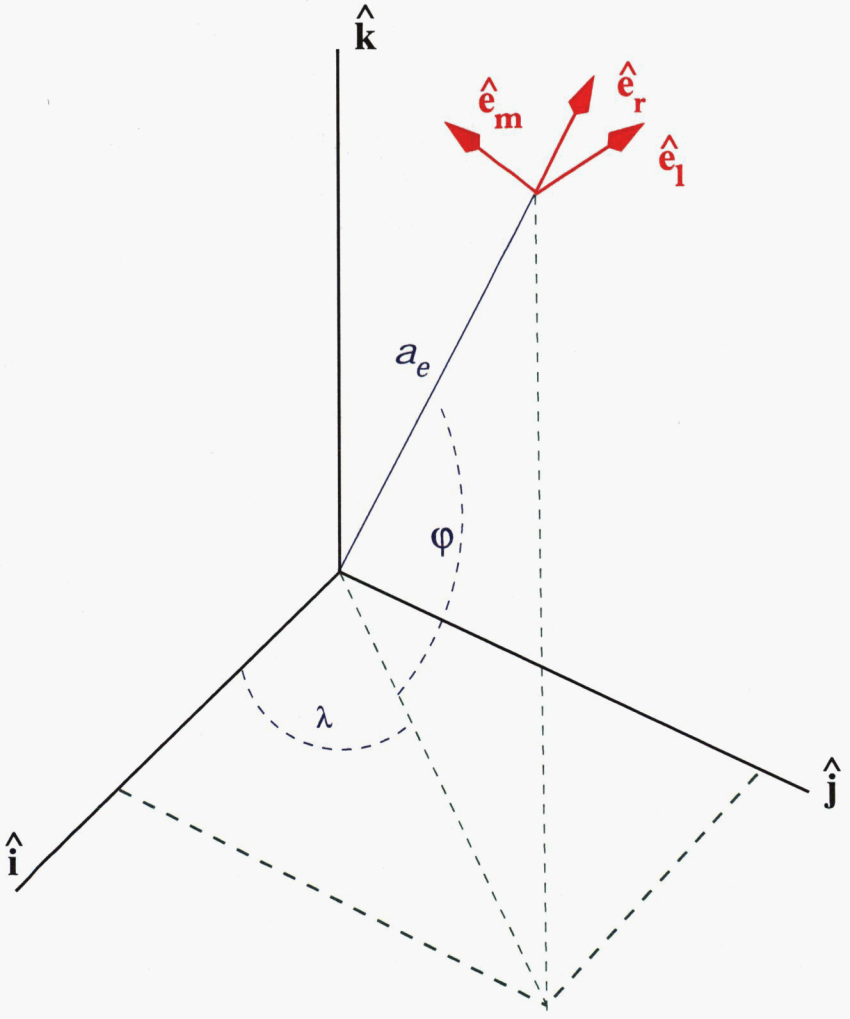


Figure 2: The spherical coordinate system used in the PSAS.

- Correlated observation error standard deviations σ_{oc}.
- The set of input attributes, after:
  - observations not used in the analysis have been eliminated
  - they have been sorted
  - their number has been reduced by superobbing (discussed in Section 6.4).

During the analysis, the PSAS calculates *derived observation attributes*:

- Sounding index **ks** (`INTEGER`), corresponding to fixed (φ, λ).
- Correlated and uncorrelated observation error standard deviations σ_{oc} and σ_{ou}, respectively (`REAL`).
- Forecast error standard deviations at observation locations σ_f.
- Cartesian components of the unit vectors $\hat{\mathbf{e}}_r$, $\hat{\mathbf{e}}_l$, and $\hat{\mathbf{e}}_m$, evaluated at observation locations (`REAL`).

- Observation elimination flag `kl` (`LOGICAL`).
- Vertical and latitudinal indices to covariance data look-up tables used to compute HP^fH^T, R, and P^fH^T (`INTEGER`).

All the above attributes are represented by p-vectors. The OMF values lie on an unstructured grid in the so-called *observation space*, with a typical distribution corresponding to six hours of observations shown in Figure 3. The mapping of observational attribute vectors to memory is described below—see Figure 5 below. In what follows, the expression "pointer" refers to an integer index into an array, e.g., of observations or their attributes.

The PSAS partitions the sphere into `maxreg` non-overlapping regions. An icosahedral decomposition (Figure 4) with `maxreg` = 80 is used v1.5.1 [Pfaendtner, 1996]. The regions are labeled by the index `kr` from 1 to `maxreg`, where `maxreg` is defined in the include file `maxreg.h`. Each of the above attribute p-vectors is sorted into `maxreg` regional segments numbered `kr`, according to which region number the observation is located in. Given `kr` from the set of {1,2,...,`maxreg`}, the integer `iregbeg(kr)` points to the beginning of region `kr`, and `ireglen(kr)` is the number of observations in region `kr`. Thus, `iregbeg(kr)` + `ireglen(kr)` −1 points to the end of region `kr`. The points in region `kr` are then sorted by data type `kt`, yielding a set of `kr/kt` *segments*. Given `kr` from the set of {1,2,...,`maxreg`} of regions, and `kt` from the set of {1,2,...,`ktmax`} of data types, the integer `itypbeg(kt,kr)` points to the beginning of the data type segment region `kt` in region `kr`, and `ityplen(kt,kr)` is the number of points in this segment. The index `itypbeg(kt,kr)` is rarely used, since for `kt` = 1, `itypbeg(1,kr)` = `iregbeg(kr)`, and for `kt` one of {2, ..., `ktmax`},

$$\texttt{itypbeg}(\texttt{kt},\texttt{kr}) = \texttt{iregbeg}(\texttt{kr}) + \sum_{i=1}^{\texttt{kt}-1} \texttt{ityplen}(i,\texttt{kr}).$$

Each of these `kr/kt` segments is then sorted in lexicographic order by data source index `kx`, latitude, longitude, and level (Figure 5). No indexing arrays are used for the lexicographic sorting scheme.

3.1.3 The Regular Latitude/Longitude Grid

The PSAS reads in forecast error standard deviations on a regular three-dimensional latitude/longitude grid with vertical pressure coordinate. The AI that the PSAS calculates are also output on this grid. The grid dimensions are defined in the include file `gridxx.h`, and are currently:

- `jdimx`: the number of latitude bands. Currently `jdimx` = 91, and $\Delta\varphi$ =2°. For `j` = 1, ..., `jdimx` the latitude $\varphi_{\texttt{j}}$ for band `j` is

$$\varphi_{\texttt{j}} = -90. + (\texttt{j} - 1)\Delta\varphi. \tag{4}$$

- `idimx`: the number of longitude bands. Currently `idimx` = 144, and $\Delta\lambda = 2.5°$. For `i` = 1, ..., `idimx` the longitude $\lambda_{\texttt{i}}$ is

$$\lambda_{\texttt{i}} = -180. + (\texttt{i} - 1)\Delta\lambda. \tag{5}$$

- `kdimx`: the maximum number of vertical levels. Currently `kdimx` = 29, with these 29 pressure levels ranging from 1000 hPa to 0.01 hPa.

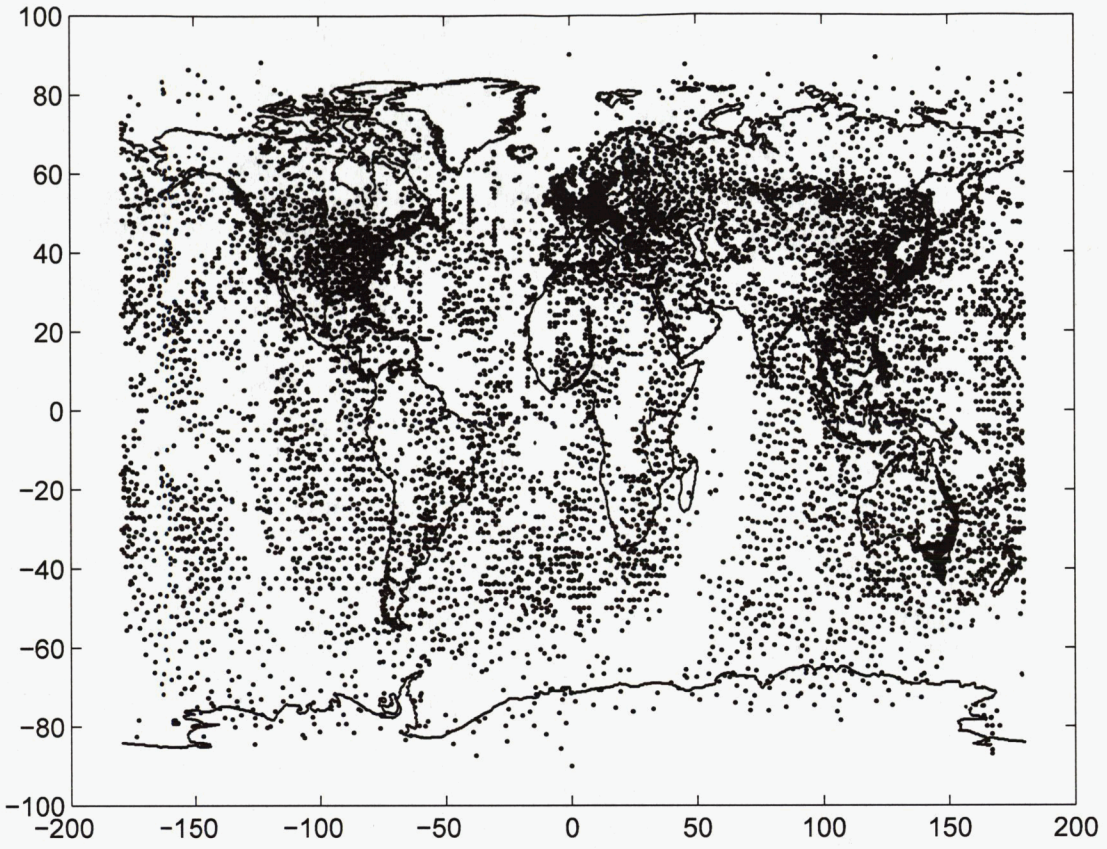


Figure 3: Typical geographic distribution of observations over a six-hour interval that are input into the PSAS.

The number of pressure levels on which the upper-air analyses are performed, and level values are read from the PSAS resource file. A complete description of the PSAS resource file is presented in the PSAS Users' Guide [da Silva et al., 1999]. Sea-level pressure is set in the include file `pres4slp.h`, and defined by the parameter `pres4slp`. Currently, `pres4slp` = 1000 hPa.

3.1.4 The Quasi-Equal Area Grid

PSAS calculates analysis increments by applying Eqn. (2) to the solution $\mathbf{x}$ of Eqn. (1). This transformation is from observation space to a quasi-equal area (QEA) grid whose vertical levels correspond to the analysis levels. AI values are then horizontally interpolated from the QEA grid to the regular 2° by 2.5° analysis grid. In this section we discuss the horizontal distribution of the QEA gridpoints.

The horizontal QEA grid has the following properties:

- The number of latitude bands is `jdimx`, with the latitude values $\varphi_1, \ldots, \varphi_{\texttt{jdimx}}$ defined in Eqn. (4).
- The QEA and analysis gridpoints coincide for latitudes $|\varphi| \leq$ `eaytresh`, where `eaytresh`

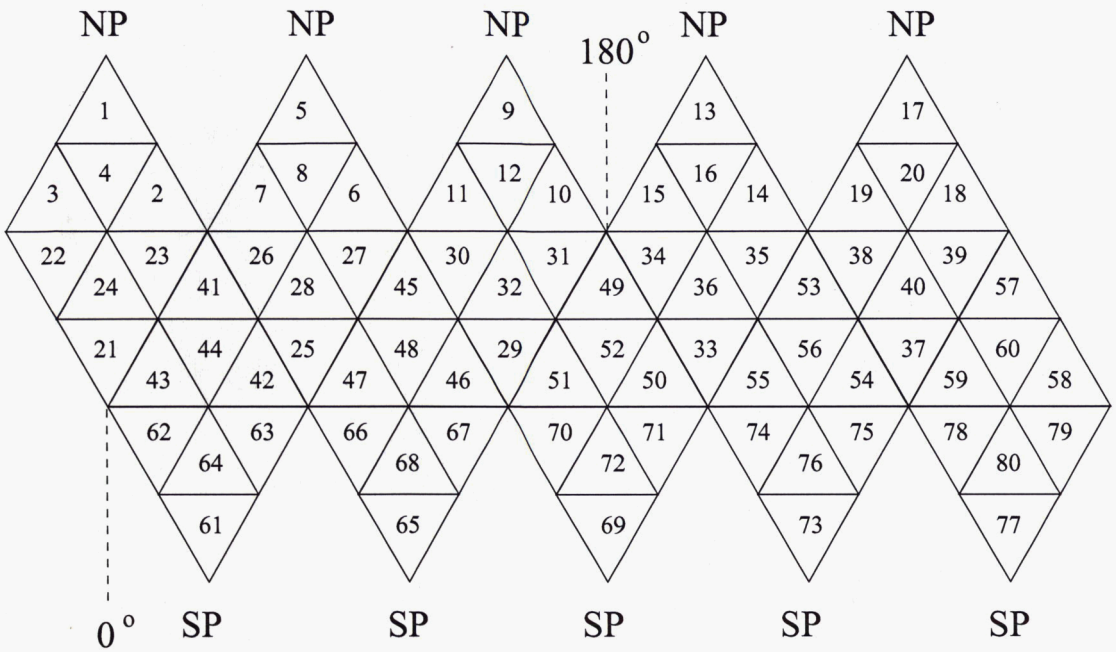


Figure 4: Icosahedral decomposition used in the PSAS, with region numbering.

is a threshold, input at runtime from the PSAS resource file. These latitude bands have index `j` in the set {`j_south`, ..., `j_north`}.

- QEA and analysis gridpoints in the latitude bands at the North and South poles coincide. These latitude bands have index `j` = 1 and `j` = `jdimx` for the South and North poles, respectively.

- For latitude bands with index j one of $\{2, \ldots,$ `j_south` $- 1\}$ or $\{$`j_north` $+ 1, \ldots,$ `jdimx` $- 1\}$, the number of longitude points `nlon` is

$$\texttt{nlon} = \texttt{nint}(\texttt{npole} + (\texttt{idimx} - \texttt{npole}) \cos \varphi_j), \tag{6}$$

  where `nint()` is the nearest integer function, `idimx` is as in eqn. (4), and `npole` is a parameter defined (with value `npole` = 4) in the include file `qea.h`. The longitude values in the band have equal spacing `dlon` = `360.0/nlon`, and for `i` = 1, ..., `nlon`,

$$\lambda_{\texttt{i}} = -180. + (\texttt{i} - 1) * \texttt{dlon}.$$

The QEA grid parameters and are stored in the common blocks `gridprm` and `gridarr`, defined in the include file `qea.h`:

```
common / gridprm / eaytresh, j_south, j_north
common / gridarr / lea_beg(jdimx),lea_len(jdimx),ea_lon(idimx*jdimx)
```

The horizontal QEA grid is initialized using the following procedure:

1. The latitude threshold `eaytresh` is read from the PSAS resource file by the routine `gridxx0()`.

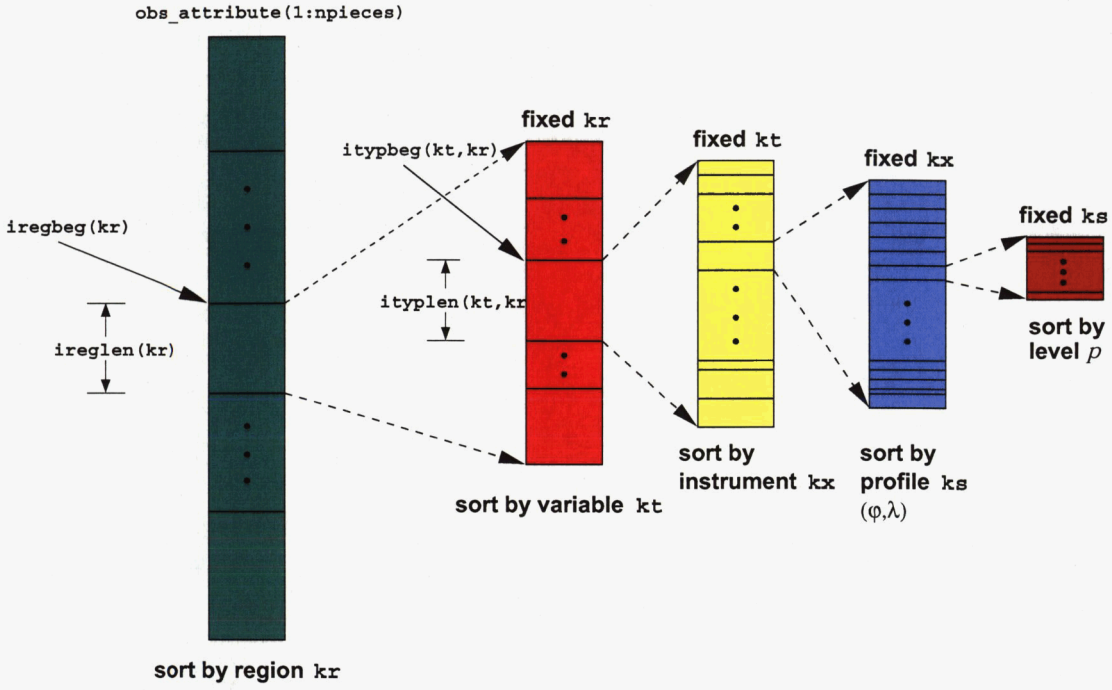


Figure 5: Regional decomposition and sorting hierarchy that is applied to each observation attribute vector.

2. The routine `eagrid()` uses the value of `eaytresh` to calculate the number `ngrid` of horizontal QEA gridpoints, and calculates their longitude values, storing them in the array `ea_lon`. The array `ea_lon` is indexed by the arrays `lea_beg(1:jdimx)` and `lea_len(1:jdimx)`, which define latitude band starting indices and lengths, respectively (Figure 6).

The construction of the multivariate three-dimensional QEA grid vector used in Eqn. (2) is described in Section 9.

3.2 Block Decomposition of Matrices

The two level `kr/kt` segment decomposition applied to the data vectors (described in Sections 3.1.2 and 3.1.4) is used to block decompose the matrices in Eqns. (1) and (2) (Figure 7). The first, coarser level of decomposition of the rows and columns is based on the regional decomposition of the attribute vectors. This yields a set of $\mathtt{maxreg}^2$ blocks. The black grid in Figure 7 represents regional block boundaries. The rows and columns of these regional blocks are sorted by `kt` as in Figure 5. This sorting by `kr` followed by `kt` yields a set of $\mathtt{maxreg}^2\mathtt{ktmax}^2$ `kr/kt` *matrix blocks*. The white lines in Figure 7 represent boundaries of `kr/kt` blocks. Each `kr/kt` block is uniquely specified by row and column values of `kr` and `kt` by $(\mathtt{kr}_i, \mathtt{kt}_i, \mathtt{kr}_j, \mathtt{kt}_j)$, where $i = 1, \ldots, \mathtt{maxreg} * \mathtt{ktmax}$ and $j = 1, \ldots, \mathtt{maxreg} * \mathtt{ktmax}$ are the `row` and `column` indices, respectively. The dimensions of the $(i,j)^{\text{th}}$ `kr/kt` block are:

$$\begin{aligned} \mathtt{ityplen}(\mathtt{kt}_i, \mathtt{kr}_i) &\quad \text{rows} \\ \mathtt{ityplen}(\mathtt{kt}_j, \mathtt{kr}_j) &\quad \text{columns} \\ \mathtt{ityplen}(\mathtt{kt}_i, \mathtt{kr}_i) * \mathtt{ityplen}(\mathtt{kt}_j, \mathtt{kr}_j) &\quad \text{elements.} \end{aligned}$$

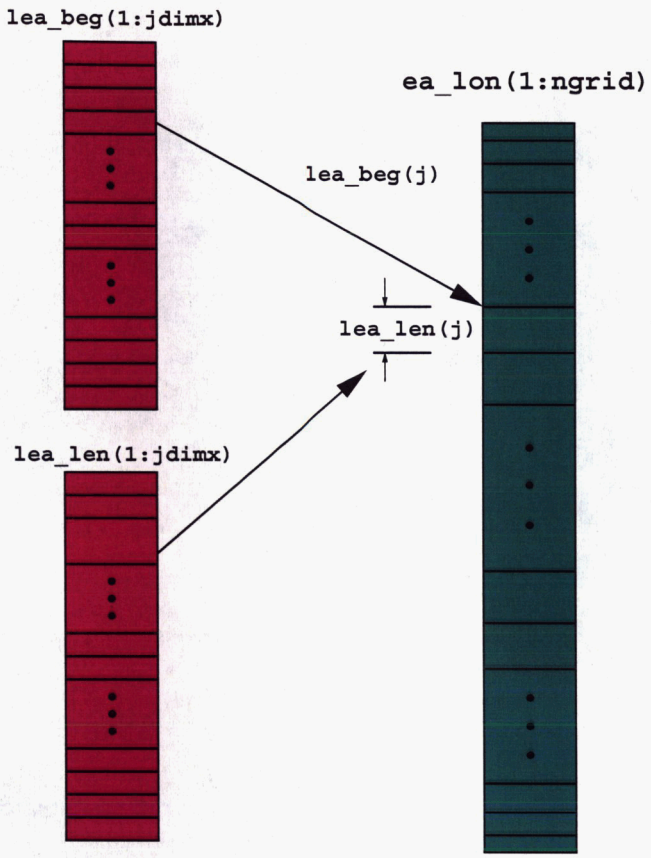


Figure 6: QEA horizontal grid longitude vector and indexing arrays.

Sparsity is introduced using compactly supported error covariance models [Gaspari and Cohn, 1999]. In the the PSAS v1.5.1, error covariance matrix blocks for regions separated by distances greater than 6030km are set to zero, and thus are not calculated. This assumption renders the matrices used in the multivariate upper-air analysis C^h, C^ψ, and C^χ defined below approximately 40% full.

3.3 Operator Formulation of PSAS

The PSAS solves Eqns. (1-2) without explicitly forming the matrices HP^fH^T, R, and P^fH^T: these matrices are represented using the *factored-operator formulation* described in [Guo et al., 1998]. The matrix-vector products $HP^fH^T\mathbf{x}$, $R\mathbf{x}$, and $P^fH^T\mathbf{x}$ are implemented as succession of operators acting on the vector $\mathbf{x}$. This approach reduces the computational complexity compared with the direct computation of the matrices HP^fH^T, R, and P^fH^T.

The factored-operator formulation of the $p_{\mathbf{kt}} \times p_{\mathbf{kt}}$ matrix HP^fH^T for the upper-air height-wind analysis is written as

$$\begin{aligned} HP^fH^T &= \Gamma^h\Sigma^hC^h\Sigma^{hT}\Gamma^{hT} + \Gamma^\psi\Sigma^\psi C^\psi\Sigma^{\psi T}\Gamma^{\psi T} \\ &+ \Gamma^\chi\Sigma^\chi C^\chi\Sigma^{\chi T}\Gamma^{\chi T} \end{aligned} , \tag{7}$$

where $p_{\mathbf{kt}}$ is the number of observations in the upper-air height-wind analysis. The first

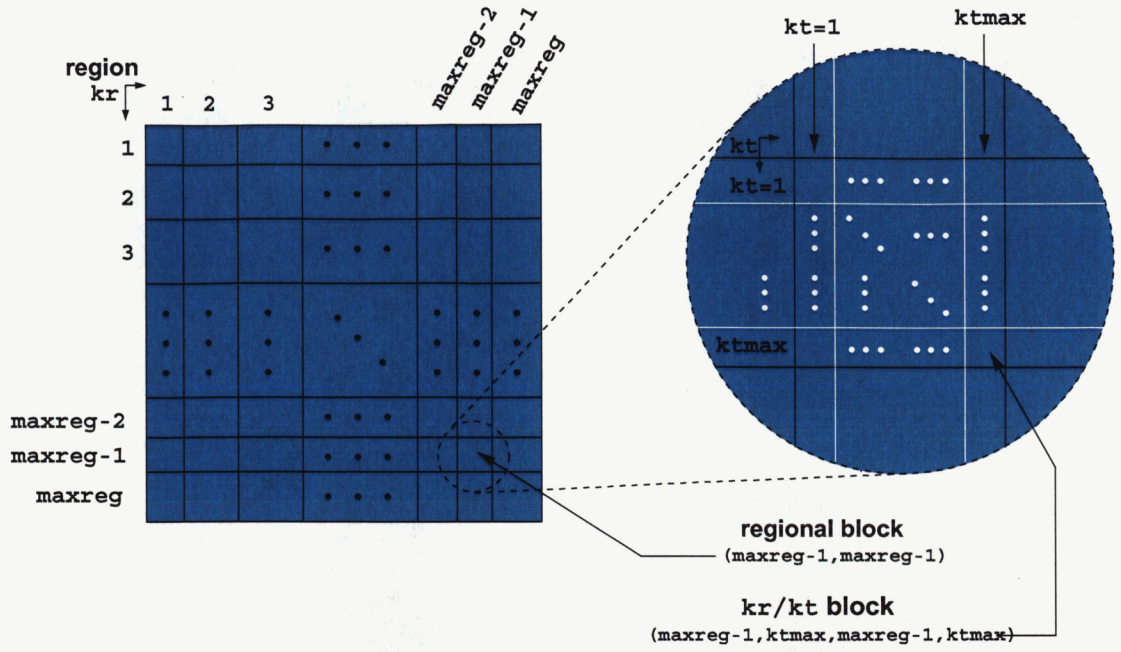


Figure 7: Block decomposition of matrices in the PSAS.

term in Eqn. (7) includes the multivariate upper-air height h and wind (u, v) forecast error covariances. Appendix C of [Guo et al., 1998] shows a similar expression for the multivariate sea-level pressure p_{sl} and wind (u_{sl}, v_{sl}) forecast error covariances. The second term in Eqn. (7) is the multivariate upper-air wind and multivariate sea-level wind forecast error covariances derived from forecast errors modeled using a streamfunction ψ. The third term in Eqn. (7) is the multivariate upper-air wind and multivariate sea-level wind forecast error covariances derived from forecast errors modeled using a velocity potential χ.

The univariate water mixing ratio forecast error covariance matrix is written

$$HP^f H^T = \Sigma^q C^q \Sigma^{qT}. \tag{8}$$

The factored-operator formulation of the observation error covariance operator R is

$$\begin{aligned} R &= R_c + R_u \\ &= \Sigma_c^o C_c^o \Sigma_c^o + \Sigma_u^o C_u^o \Sigma_u^o. \end{aligned} \tag{9}$$

The first and second terms in Eqn. (9) are the *horizontally correlated* and *horizontally uncorrelated* observation error covariances, respectively.

Each of the factors in the terms in Eqn. (7) are $p_{\mathbf{kt}} \times p_{\mathbf{kt}}$ matrices, and the ordering of their rows and columns is defined by the **kr/kt** decomposition and lexicographic sorting scheme shown in Figure 5. This sorting scheme, and the definitions of the operator factors from [Guo et al., 1998] have the following **implications**:

1. The factor Γ^h is **kr/kt** block-diagonal, and has three nonzero bands in each of the diagonal **kr/kt** block. Multiplication of a $p_{\mathbf{kt}}$-vector by Γ^h requires $\mathcal{O}(p_{\mathbf{kt}})$ operations.

2. Γ^ψ and Γ^χ in Eqn. (7) are **kr/kt** block-diagonal, and each have two nonzero bands. Multiplication of a $p_{\mathbf{kt}}$-vector by Γ^ψ or Γ^χ requires $\mathcal{O}(p_{\mathbf{kt}})$ operations.

Table 1: Data type index **kt** values

Variable	kt-value
Sea-level zonal wind u_{sl}	1
Sea-level meridional wind v_{sl}	2
Sea-level pressure p_{sl}	3
Upper-air zonal wind u	4
Upper-air meridional wind u	5
Upper-air geopotential height h	6
Upper-air water vapor mixing ratio q	7

3. Σ^h, Σ^ψ, and Σ^χ are diagonal. Multiplication of a $p_{\mathbf{kt}}$-vector by Σ^h, Σ^ψ, or Σ^χ requires $\mathcal{O}(p_{\mathbf{kt}})$ operations.

4. The matrices C^h, C^ψ, and C^χ are dense matrices (currently about 40% of their elements are nonzero), with block structure discussed in Section 3.2.

5. The matrices Σ_c^o and Σ_u^o are diagonal.

6. The matrix C_u^o applies only to vertically correlated observations, and thus is profile-diagonal. Multiplication of a $p_{\mathbf{kt}}$-vector by C_u^o requires $<< \mathcal{O}(p_{\mathbf{kt}}^2)$ operations.

7. The matrix C_c^o is a dense matrix. Multiplication of a $p_{\mathbf{kt}}$-vector by C_c^o requires $\mathcal{O}(p_{\mathbf{kt}}^2)$ operations.

The factored-operator formulation of the $n_{\mathbf{kt}} \times p_{\mathbf{kt}}$ matrix $P^f H^T$ for the upper-air height-wind analysis is

$$\begin{aligned} P^f H^T &= \Gamma_L^h \Sigma_L^h C^h \Sigma_R^{hT} \Gamma_R^{hT} + \Gamma_L^\psi \Sigma_L^\psi C^\psi \Sigma_R^{\psi T} \Gamma_R^{\psi T} \\ &\quad + \Gamma_L^\chi \Sigma_L^\chi C^\chi \Sigma_R^{\chi T} \Gamma_R^{\chi T}, \end{aligned} \tag{10}$$

where $n_{\mathbf{kt}}$ is the number of grid points in the analysis. The individual forecast error covariance terms in Eqn. (10) are defined as in Eqn. (7). The matrices with subscripts L and R in Eqn. (10) have dimensions $n_{\mathbf{kt}} \times n_{\mathbf{kt}}$ and $p_{\mathbf{kt}} \times p_{\mathbf{kt}}$ respectively. The matrices C^h, C^ψ, and C^χ have dimension $n_{\mathbf{kt}} \times p_{\mathbf{kt}}$. The above implications numbered 1 and 2 apply to the $p_{\mathbf{kt}} \times p_{\mathbf{kt}}$ matrices, multiplication of an $n_{\mathbf{kt}}$-vector by each of the $n_{\mathbf{kt}} \times n_{\mathbf{kt}}$ matrices in Eqn. (10) requires $\mathcal{O}(n_{\mathbf{kt}})$ operations, and the multiplication of an $p_{\mathbf{kt}}$-vector by each of the $n_{\mathbf{kt}} \times p_{\mathbf{kt}}$ matrices in Eqn. (10) requires $\mathcal{O}(n_{\mathbf{kt}} p_{\mathbf{kt}})$ operations.

The definitions of the **kt** values used in these tables are given in Table 1. The non-zero elements of Γ^h, Γ^ψ, Γ^χ, Σ^h, Σ^ψ, Σ^χ, C^h, C^ψ, and C^χ are shown in Tables 2 – 10. We use notation that is consistent with [Guo et al., 1998].

Table 2: Nonzero elements of the multivariate operator Γ^h

Row kt-value $\mathtt{kt}_i$	**Column kt-value** $\mathtt{kt}_j$	**Value**
1	1	$\alpha_{um}(\varphi, p_{sl})$
1	2	$\alpha_{ul}(\varphi, p_{sl})$
2	1	$\alpha_{vm}(\varphi, p_{sl})$
2	2	$\alpha_{vl}(\varphi, p_{sl})$
3	3	1
4	4	$\alpha_{um}(\varphi, p)$
4	5	$\alpha_{ul}(\varphi, p)$
5	4	$\alpha_{vm}(\varphi, p)$
5	5	$\alpha_{vl}(\varphi, p)$
6	6	1
7	7	1

Table 3: Nonzero elements of the multivariate operator Γ^ψ

Row kt-value $\mathtt{kt}_i$	**Column kt-value** $\mathtt{kt}_j$	**Value**
1	1	-1
2	2	1
4	4	-1
5	5	1

Table 4: Nonzero elements of the multivariate operator Γ^χ

Row kt-value $\mathtt{kt}_i$	**Column kt-value** $\mathtt{kt}_j$	**Value**
1	2	1
2	1	1
4	5	1
5	4	1

Table 5: Elements of the diagonal matrix Σ^h ($\sigma^{p_{sl}} = g\bar{\rho}\sigma^h$)

kt-value	**Value**
1	$\frac{\sigma^{p_{sl}}\sqrt{\rho^{h'}(1)}}{2\Omega\bar{\rho}a}$
2	$\frac{\sigma^{p_{sl}}\sqrt{\rho^{h'}(1)}}{2\Omega\bar{\rho}a}$
3	$\sigma^{p_{sl}}$
4	$\frac{g\sigma^h\sqrt{\rho^{h'}(1)}}{2\Omega a}$
5	$\frac{g\sigma^h\sqrt{\rho^{h'}(1)}}{2\Omega a}$
6	σ^h
7	σ^q

Table 6: Elements of the diagonal matrix Σ^ψ

kt-value	**Value**
1	σ^ψ
2	σ^ψ
3	0
4	σ^ψ
5	σ^ψ
6	0
7	0

Table 7: Elements of the diagonal matrix Σ^χ

kt-value	**Value**
1	σ^χ
2	σ^χ
3	0
4	σ^χ
5	σ^χ
6	0
7	0

Table 8: Nonzero elements of the operator C^h

Row kt-value $\mathtt{kt}_i$	**Column kt-value** $\mathtt{kt}_j$	**Value**
1	1	c^h_{mm}
1	2	c^h_{ml}
1	3	c^h_{mh}
2	1	c^h_{lm}
2	2	c^h_{ll}
2	3	c^h_{lh}
3	1	c^h_{hm}
3	2	c^h_{hl}
3	3	c^h_{hh}
4	4	c^h_{mm}
4	5	c^h_{ml}
4	6	c^h_{mh}
5	4	c^h_{lm}
5	5	c^h_{ll}
5	6	c^h_{lh}
6	4	c^h_{hm}
6	5	c^h_{hl}
6	6	c^h_{hh}
7	7	c^q

Table 9: Nonzero elements of the operator C^ψ

Row kt-value $\mathtt{kt}_i$	**Column kt-value** $\mathtt{kt}_j$	**Value**
1	1	c^ψ_{mm}
1	2	c^ψ_{ml}
2	1	c^ψ_{lm}
2	2	c^ψ_{ll}
4	4	c^ψ_{mm}
4	5	c^ψ_{ml}
5	4	c^ψ_{lm}
5	5	c^ψ_{ll}

Table 10: Nonzero elements of the operator C^{χ}

Row kt-value $\mathtt{kt}_i$	Column kt-value $\mathtt{kt}_j$	Value
1	1	c^{χ}_{mm}
1	2	c^{χ}_{ml}
2	1	c^{χ}_{lm}
2	2	c^{χ}_{ll}
4	4	c^{χ}_{mm}
4	5	c^{χ}_{ml}
5	4	c^{χ}_{lm}
5	5	c^{χ}_{ll}

3.4 Data Management in PSAS

3.4.1 PSAS Data Flow

The analysis and library interface routines in PSAS have the following characteristics:

- Input from and output to routines calling the PSAS analysis and library interfaces is via the interface routines of the PSAS.
- Apart from the data structures in the subroutine interfaces of the PSAS, necessary data is input to the PSAS from either the PSAS resource file (Section 3.4.3), or a data file defined by a PSAS resource.
- Diagnostic output is directed to `stdout`.
- Parameter data used to calculate error covariance operators is stored in data structures defined within modules. Often, these data structures then enter routines via `USE` statements, though sometimes they are passed to lower-level routines via interfaces.
- Some parameter data is stored in `COMMON` blocks defined in include files.

3.4.2 The Covariance Data Life Cycle

Calculation of operators in Eqns. (7-10) requires the following quantities:

- Horizontally correlated and horizontally uncorrelated observation error standard deviations σ_{oc} and σ_{ou}.
- Observation error horizontal correlations ρ_{oc}.
- Horizontally correlated and horizontally uncorrelated observation error vertical correlations ν_{oc} and and ν_{ou}.
- Forecast error standard deviations σ^h, $\sigma^{p_{sl}}$, σ^q, σ^{ψ}, and σ^{χ}.
- Elements of the coupled mass-balance operator for forecast height/wind errors α_{um}, α_{vm}, α_{ul}, and α_{vl}.

- Forecast error vertical correlations .
- Forecast error horizontal correlations.

These quantities progress through the *covariance data life cycle* shown in Figure 8. In this cycle the covariance data exist in a series of *states*, and each state is the result of a processing *stage*.

The covariance data states are:

- **State I**: Variables representing resources. These are data input from the PSAS resource file (Section 3.4.3), and include
  - tables of observation error standard deviations.
  - tables of observation error vertical correlations.
  - level-dependent tables of parameters used to evaluate observation error horizontal correlation functions.
  - the name of the data file from which gridded forecast error standard deviations are input.
  - tables of forecast error vertical correlations.
  - level-dependent tables of parameters used to evaluate forecast error horizontal correlation functions.
  - level-dependent tables of parameters for functions used to calculate the multivariate parameters α_{um}, α_{vm}, α_{ul}, and α_{vl}.
  - level-dependent tables of parameters for functions used to model σ^{ψ} and σ^{χ}.
- **State II**: Look-up tables. These data are calculated from the State I data, and are:
  - attribute vectors of the elements of the diagonal matrices Σ^h in Eqn. (7) and Σ^h_L and Σ^h_R in Eqn. (10).
  - attribute vectors of σ_{ou} and σ_{oc}, which form the diagonals of Σ^o_u and Σ^o_c, respectively.
  - *Indirect Matrix* (IMAT) tables used to calculate:
    * Forecast error vertical correlation coefficients.
    * Forecast error horizontal correlation functions and their first and second derivatives with respect to the polarity index τ defined in Eqn. (3) above.
    * Observation error vertical correlation coefficients.
    * Observation error horizontal correlation functions.
    * Forecast error standard deviations σ^{ψ} and σ^{χ}.
    * The geostrophic balance parameters α_{um}, α_{vm}, α_{ul}, and α_{vl}.
- **State III**: Block matrix operators. These are individual `kr`/`kt` blocks of the operators in Eqns. (7-10).

The covariance data processing stages are:

- *Stage I Processing* is the parsing of input data from the PSAS resource file. This processing stage is discussed in Section 5.
- *Stage II Processing* is the refinement and tabulation of State I data, resulting in State II data. This processing stage is discussed in Section 7.
- *Stage III Processing* is use of IMAT tables and attribute vectors to calculate block matrix operators. This processing stage is discussed in Sections 8 - 10.

Detailed data life cycle diagrams for the operators in Eqns. (7-10) are presented in Appendix D.

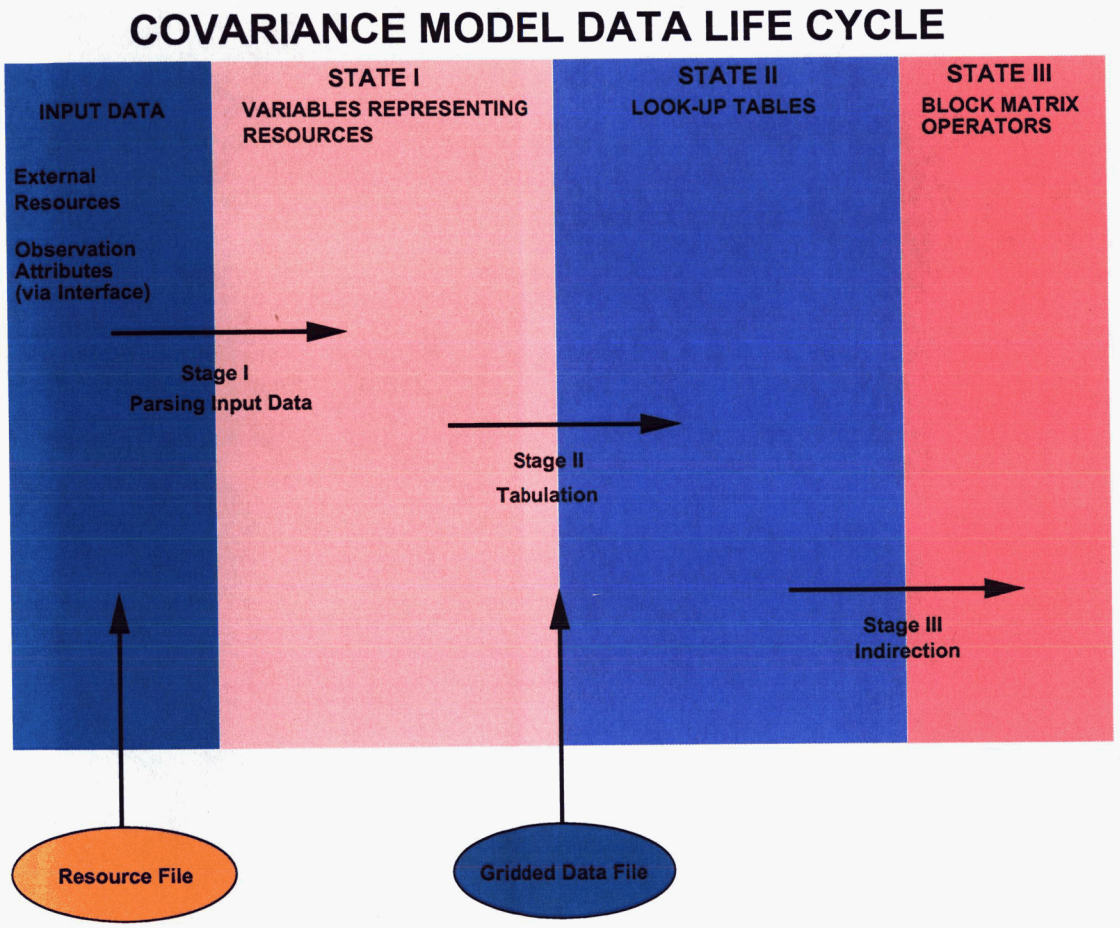


Figure 8: Data life cycle for parameters used in PSAS.

3.4.3 The PSAS Resource File

The majority of Stage II processing is the input of secondary resources from the PSAS resource file. The entire resource file (default name **psas.rc**) is read into memory, and parsed using I90 functions[1]. Data in the resource file have character *resource label*, followed by the actual data.

Example 3.4.3.1: The latitude threshold **eaytresh** at which the gridpoints on quasi-equal-area grid are no longer the default analysis grid locations has this entry in the resource file:

```
latitude_treshold_for_equal_area_grid:        45
```

The routine **gridxx0()** uses the following block of code to read the value of **eaytresh**:

```
call LABLIN ( 'latitude_treshold_for_equal_area_grid:' )
eaytresh = abs(FLTGET ( 90. ))
```

The call to **LABLIN('latitude_treshold_for_equal_area_grid:')** tells I90 to find this string in the block of memory where the resource file resides. The function call **FLTGET (90.)** means "convert the next string to a real number," in the absence of a string before the carriage return, return the default value of 90.0." A sample resource file **psas.rc** is given in Appendix C. Further description of resource files can be found in the I90 documentation and the PSAS Users' Guide [da Silva et al., 1999].

[1]The I90 library and its documentation are available at the URL `ftp://niteroi.gsfc.nasa.gov/pub/packages/i90`

4 Top Level Control Flow — `getAIall()`

4.1 Analysis Interfaces to the PSAS

The analysis increments (AI), i.e., the quantities denoted $\mathbf{w}^a - \mathbf{w}^f$ in Eqn. (1), are computed from OMF's by calling one of the routines **`getAIpuv()`**, **`getAIzuv()`**, **`getAImix()`**, and **`getAIall()`**. The routine **`getAIzuv()`** returns AI for upper air winds u and v, and upper air geopotential heights h. The routine **`getAIpuv()`** returns AI for sea-level winds u_{sl} and v_{sl}, and sea-level pressure p_{sl}. The routine **`getAImix()`** returns AI for upper-air water vapor mixing ratio q. The routine **`getAIall()`** returns AI for all the aforementioned quantities.

Since the logic in **`getAIpuv()`**, **`getAIzuv()`**, and **`getAImix()`** is contained in that of **`getAIall()`**, this section explains the routine **`getAIall()`** in detail. Appendix A contains a discussion of how analysis routines **`getAIzuv()`**, **`getAIpuv()`**, and **`getAImix()`** differ from **`getAIall()`**.

The major subroutines called by **`getAIall()`** are:

- **`solve4x()`**: Solves Eqn. (1) for $\mathbf{x}$.
- **`getAinc()`**: Uses Eqn. (2) to $\mathbf{x}$ to produce AI.

The interface to **`getAIall()`** is

```
subroutine getAIall( npieces, lat_list, lon_list, pres_list, &
                     time_list, kx_list, kt_list, dels_list, &
                     sigF_list, sigO_list,                   &
                     im, jnp, mlev, pres_lev,                &
                     psl_sigF, usl_sigF, vsl_sigF,           &
                     z_sigF, u_sigF, v_sigF,                 &
                     mix_sigF,                               &
                     psl_inc, usl_inc, vsl_inc,              &
                     z_inc, u_inc, v_inc,                    &
                     mix_inc,                                &
                     psl_sigA, usl_sigA, vsl_sigA,           &
                     z_sigA, u_sigA, v_sigA,                 &
                     mix_sigA                                 ) .
```

Table 2 specifies the arguments to **`getAIall()`**. The analysis interface routines (below **`getAIall()`**) share the same calling tree, shown in Figure 9.

In Section 4.2 we describe the variables used in **`getAIall()`**, as well as the include files and modules it uses. In Section 4.2 we trace through the execution of **`getAIall()`**, describing the function of the routines called by **`getAIall()`**.

4.2 Description of Variables and Include Files used in `getAIall()`

Some of the data used in **`getAIall()`** are defined in separate Fortran 90 modules and enter this routine via **`USE`** statements. Functions from the inpack module **`m_inpak90`** are also used in **`getAIall()`**. The variables in **`getAIall()`** are:

- **`veclats`** (**`REAL`**): IMAT latitude values. Defined in the module **`rlat_imat`**.

- **MXveclat** (**INTEGER**): maximum size of **veclats**, the array of IMAT latitudes. Defined in the module **config**.
- **nveclat** (**INTEGER**): the number of IMAT latitudes. Defined in the module **rlat_imat**.
- **plev_oe** (**REAL**): pressure levels in State I observation error statistics tables. Defined in the module **OEclass_tbl**.
- **nlev_oe** (**INTEGER**): the number of pressure levels in State I observation error statistics tables. Defined in the module **OEclass_tbl**.
- **pres4slp** (**REAL**): sea-level pressure. Defined the module **config**.
- **stdout** (**INTEGER**): Standard diagnostic output device number. Defined in the module **config**.
- **stderr** (**INTEGER**): Standard diagnostic error device number. Defined in the module **config**.

Header files included in **getAIall()** are:

- **psasrc.h**: This file contains **CHARACTER** parameters defining the software name, version, and default name of the PSAS resource file **psas.rc**. The resource file name is stored in the common block **rsrcfile**:

```
common/rsrcfile/psasrc
```

- **ktmax.h**: The number of defined values of the variable index **kt** is the **INTEGER** parameter **ktmax**. The **kt** index values for the variables u_{sl}, v_{sl}, p_{sl}, u, v, h, and q are given by **INTEGER** parameters **ktus**, **ktvs**, **ktslp**, **ktuu**, **ktvv**, **ktHH**, and **ktqq**, respectively.
- **kxmax.h**: The number of defined values of the data source index **kx** is the **INTEGER** parameter **kxmax**.
- **ktmax.h**: The number of defined values of the data type index **kt** is the **INTEGER** parameter **ktmax**.
- **ktwanted.h**: the **LOGICAL** control array **ktwanted**, controlling the data types for which AIs are computed. This variable resides in the common block **ktcontrl**:

```
common /ktcontrl/ ktwanted(ktmax)
```

- **lvmax.h**: The maximum number of vertical analysis levels is determined by the parameter **lvmax**. The maximum number of IMAT vertical levels **MXveclev**.
- **levtabl.h**: the number **nveclev** of IMAT vertical levels, and IMAT vertical level values **pveclev**. Both variables reside in the common block **levtabl**:

```
common/levtabl/ nveclev, pveclev(MXveclev)
```

- **maxreg.h**: the maximum number of regions used in the **kr**-decomposition of the observation attribute vectors, defined by the **INTEGER** parameter **maxreg**.
- **bands.h**: variables and common blocks used by the conjugate gradient solver.

Some of the workspace used by **getAIall()** is dynamically allocated:

- **sigU_list(npieces)**: Horizontally uncorrelated observation error standard deviations σ_{ou} (**REAL**).
- **xvec(npieces)**: The intermediate vector **x** (**REAL**).
- **kl(npieces)**: A **LOGICAL** flag array that determines whether or not certain entries in observation attribute arrays are to be used in the analysis.

4.3 Control Flow through getAIall()

4.3.1 Stage I Processing of Covariance Data

The first execution of **getAIall()** is a unix call to **getenv()** to load the shell environment variable **PSASRC**, if defined. If no value of **PSASRC** is defined, the default resource file name defined in **psasrc.h** is used. This resource file, tagged by the **CHARACTER** variable **psasrc**, is loaded into memory by a call to the inpak routine **I90_LoadF()**.

The routine **initRSRC** is called to initialize run-time data:

- Data type information, including data type names, data type units, data type descriptions, and a table of data type dependencies in multivariate covariance models.
- Data source information, including observation error class, data source rank, and data source description .
- State I covariance data tables.
- Conjugate gradient solver control parameters.
- Icosahedral region definitions and names.
- Control parameters for the proximity elimination package **proxel()**.
- Tabulated values of trigonometric functions used by subroutine **qtrig()**.
- Values of data type index **kt** for which AI's are to be computed.

The variable **pres_list(n)** = **pres4slp** for each **n** = 1,...,**npieces** where **kt_list(n)** is one of **ktslp**, **ktus**, or **ktvs**.

Root-mean-square observation error standard deviation (correlated and uncorrelated) are log-linearly interpolated to observation locations from the tables **sigOu** and **sigOc** in the routine **intp_sigO()**:

```
call intp_sigO(npieces,kx_list,kt_list,pres_list,sigO_list,sigU_list)
```

The tables **sigOu** and **sigOc** are defined in the module **OEclas_tbl**. The interpolated **sigOu** is returned in **sigU_list(1:npieces)** and the interpolated **sigOc** is returned in **sigO_list(1:npieces)**.

Table 2: Data passed into `getAIall()` via its interface.

Variable	Type	Intent	Description
npieces	INTEGER	INOUT	Number of Observations
lat_list	REAL(npieces)	INOUT	Latitude
lon_list	REAL(npieces)	INOUT	Longitude
pres_list	REAL(npieces)	INOUT	Pressure (hPa)
time_list	REAL(npieces)	INOUT	Δt (min) from Analysis Time
kx_list	INTEGER(npieces)	INOUT	Data Source Index **kx**
kt_list	INTEGER(npieces)	INOUT	Data Type Index **kt**
dels_list	REAL(npieces)	INOUT	$\mathbf{w}^o - H\mathbf{w}^f$
sigF_list	REAL(npieces)	INOUT	Forecast Error Variances σ_f at obs. locations
sigO_list	REAL(npieces)	INOUT	Observation Error Variances σ_o
im	INTEGER	IN	Number of Longitudinal Grid Points
jnp	INTEGER	IN	Number of Latitudinal Grid Points
mlev	INTEGER	IN	Number of Grid Vertical Levels
pres_lev	REAL(mlev)	INOUT	Analysis Pressure Levels
psl_sigF	REAL(im,jnp)	IN	Forecast Sea-Level Pressure Error Variances
usl_sigF	REAL(im,jnp)	IN	Forecast u_{sl} Error Variances
vsl_sigF	REAL(im,jnp)	IN	Forecast v_{sl} Error Variances
z_sigF	REAL(im,jnp)	IN	Forecast Geopotential Height Error Variances
u_sigF	REAL(im,jnp)	IN	Forecast u Error Variances
v_sigF	REAL(im,jnp)	IN	Forecast v Error Variances
mix_sigF	REAL(im,jnp)	IN	Forecast Mixing Ratio Error Variances
psl_inc	REAL(im,jnp)	OUT	Sea-Level Pressure Analysis Increments
usl_inc	REAL(im,jnp)	OUT	u_{sl} Analysis Increments
vsl_inc	REAL(im,jnp)	OUT	v_{sl} Analysis Increments
z_inc	REAL(im,jnp)	OUT	Geopotential Height Analysis Increments
u_inc	REAL(im,jnp)	OUT	u Analysis Increments
v_inc	REAL(im,jnp)	OUT	v Analysis Increments
mix_inc	REAL(im,jnp)	OUT	Mixing Ratio Analysis Increments
psl_sigA	REAL(im,jnp)	OUT	Sea-Level Pressure Analysis Error Variances
usl_sigA	REAL(im,jnp)	OUT	u_{sl} Analysis Error Variances
vsl_sigA	REAL(im,jnp)	OUT	v_{sl} Analysis Error Variances
z_sigA	REAL(im,jnp)	OUT	Geopotential Height Analysis Error Variances
u_sigA	REAL(im,jnp)	OUT	u Analysis Error Variances
v_sigA	REAL(im,jnp)	OUT	v Analysis Error Variances
mix_sigA	REAL(im,jnp)	OUT	Mixing Ratio Analysis Error Variances

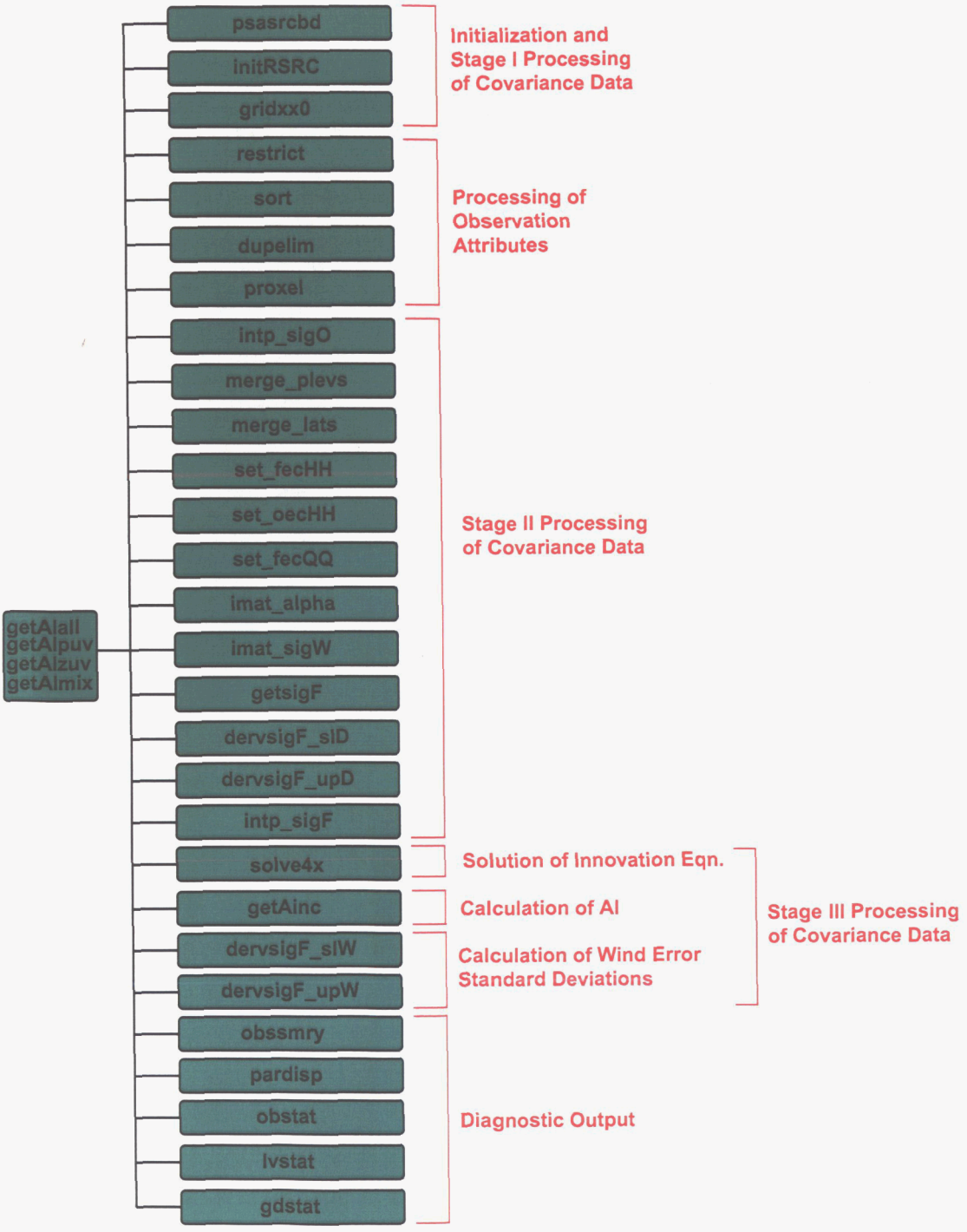


Figure 9: Top-level control flow below analysis interfaces `getAIall()`.

4.3.2 Processing of Observational Data

A call to the routine **restrict()** selects the observations used to calculate the analysis increments.

```
call restrict ( verbose,stdout,npieces,prtdat1,                &
                lat_list,lon_list,pres_list,kx_list,kt_list, &
                dels_list,sigU_list,sig0_list,sigF_list,     &
                time_list,nnobs                               )
```

Vectors of length **npieces** enter **restrict()**. The routine **restrict()** selects observations whose attributes fall within **analysis boxes** (defined in Section 6.1), or are otherwise needed for analysis. Using a call to **tofront()**, **restrict** sorts the attribute arrays so that their first **nnobs** entries are the values to be used in subsequent analysis. Further discussion of **restrict()** can be found in Section 6.1.

The value of **npieces** is set to **nnobs** for subsequent processing.

Using a call to subroutine **sort()**, the observation attribute arrays are sorted as in Section 3.1.2 and illustrated in Figure 5:

```
call sort ( myname,verbose,stdout,npieces,lat_list,lon_list, &
            pres_list,kx_list,kt_list,dels_list,sigU_list,   &
            sig0_list,sigF_list,time_list,maxreg,ktmax,      &
            iregbeg,ireglen,ityplen                           )
```

Further discussion of the routine **sort()** is given in Section 6.2.

The sorted attribute arrays are are scanned for duplicate observations. These duplicates are eliminated from the analysis by the routine **dupelim()**:

```
call dupelim (verbose,stdout,npieces,kx_list,kt_list,kl, &
              lat_list,lon_list,pres_list,dels_list,      &
              sigU_list,sig0_list,sigF_list,time_list,    &
              maxreg,iregbeg,ireglen,ktmax,ityplen         )
```

The **LOGICAL** array **kl(1:npieces)** contains flags determining data used in the analysis. If for some index **i**, **kl(i)** = **.TRUE.**, entries in the other attribute arrays with index **i** are used. The value of **npieces** is changed by **dupelim()** to the number of distinct observations. Using a call to **tofront()**, **dupelim()** sorts the attribute arrays so that their first **npieces** entries are the values to be used in subsequent analysis. Further discussion of the routine **dupelim()** is given in Section 6.3.

The next step is to reduce the number of observations through superobbing. This is implemented as follows using the proximity elimination routine **proxel()**:

```
nprox = 0
n = 1
do while(n.eq.1 .or. nprox.ne.0.and.n.le.5)
   call proxel (verbose,stdout,npieces,kx_list,kt_list,kl, &
                lat_list,lon_list,pres_list,dels_list,     &
                sig0_list,sigU_list,sigF_list,time_list,   &
```

```
                    maxreg,iregbeg,ireglen,ktmax,ityplen,nprox  )
    n=n+1
  end do
```

The number of observations eliminated in one call to `proxel()` is `nprox`. Using the routine `tofront()`, `proxel()` sorts the attribute so that their first `npieces-nprox` entries are the values to be used in subsequent analysis. Each time `proxel()` returns, the value of the variable `npieces` is reduced by `nprox`, so that `npieces` is the number of remaining observations, including the super-ob values. The attribute indexing arrays `iregbeg(1:maxreg)`, `ireglen(1:maxreg)`, and `ityplen(1:ktmax,1:maxreg)` are modified accordingly. Further discussion of the superobbing process is presented in Section 6.4.

QEA gridpoint locations are calculated by a call to `gridxx0()`. The QEA grid parameters and gridpoint data structures were described in Section 3.1.4. The call to `gridxx0()` reads in the `REAL` QEA latitude threshold `eaytresh` from the resource file.

4.3.3 Stage II Processing of Covariance Data

The initialization of the IMAT structures for the forecast and observation error covariances now occurs. The IMAT pressure levels and latitude values must first be determined. The set of IMAT pressure levels is the union of the sea-level pressure `pres4slp`, the analysis levels `pres_lev(1:mlev)`, and the observation pressure level values `pres_list(1:npieces)`. This union is found through a pair of calls to the routine `merge_plevs()`:

```
call merg_plevs(mlev,pres_lev,1,pres4slp,MXveclev,nveclev,pveclev)

call merg_plevs(npieces,pres_list,nveclev,pveclev,MXveclev,nveclev,pveclev)
```

The first call to `merg_plevs()` combines the set of analysis pressure levels with the one sea-level pressure value, resulting in the combined set of `nveclev` levels in `pveclev(1:nveclev)`. This is done by eliminating redundant pressure level values. The second call combines this set of `nveclev` levels stored in `pveclev` with the set of observation pressure level values. Upon return from this call to `merge_plevs()`, the values of `nveclev` and `pveclev` are now for the complete union set. If the union contains more than `MXveclev` elements, a subset of levels is determined by `merge_plevs()`. Further discussion of `merge_plevs()` is presented in Section 7.2.1.

Next, the `jnp` analysis grid latitude values are merged with the `npieces` observation latitude locations `lat_list(1:npieces)` to arrive at a non-redundant set of latitude table values `veclats(1:nveclat)`. This is implemented in a call to the routine `merg_lats()`.

```
call merg_lats(jnp,npieces,lat_list,MXveclat,nveclat,veclats)
```

The number of imat latitudes `nveclat` $\leq$ `MXveclat`. Further discussion of the routine `merg_lats()` is found in Section 7.2.2.

Once the IMAT pressure levels `veclevs(1:nveclev)` and latitudes `veclats(1:nveclat)` are determined, the IMAT entries can be computed. IMAT quantities calculated are:

- IMAT data for the forecast height, sea-level pressure, and wind error correlations. These data are contained in the IMATs `hfecHH`, `hfecRR`, `hfecTT`, `vfecHH`, `vfecHD`, and `vfecDD`, and calculated by a call to `set_fecHH()` (see Section 7.3.1).

- IMAT data for observation error correlations. These data are contained in the IMAT's `hoecHH` and `voecHH`, and calculated by a call to `set_oecHH()` (see Section 7.4).
- IMAT data for forecast water vapor mixing ratio error correlations. These data are contained in the IMAT's `hfecQQ` and `vfecQQ`, and calculated by a call to `set_fecQQ()` (see Section 7.3.2).
- IMAT tables of upper-air and sea-level height/wind coupled balance parameters α_{um}, α_{ul}, α_{vm}, and α_{vl}. These data are contained in the IMATs `Aum_imat`, `Aul_imat`, `Avm_imat`, and `Avl_imat`, and computed by a call to `imat_alpha()` (see Section 7.5).
- IMAT tables of forecast error standard deviations σ^{ψ} and σ^{χ}. These data are contained in the IMATs `FEsigS_imat` and `FEsigV_imat`, and are computed by a call to `imat_sigW()` (see Section 7.6).

Gridded forecast geopotential height and upper-air mixing ratio error standard deviations are read from a binary file. The gridded upper-air heights are used to calculate forecast sea-level pressure error standard deviations. These operations are performed by a call to `getsigF()` (see Section 7.7.1):

```
call getsigF(im,jnp,mlev,pres_lev,psl_sigF,z_sigF,mix_sigF)
```

Th routine `getsigF()` returns:

- Forecast sea-level pressure error standard deviations in `psl_sigF(1:im,1:jnp)`.
- Forecast upper-air height error standard deviations in `z_sigF(1:im,1:jnp,1:mlev)`.
- Forecast upper-air mixing ratio error standard deviations in `mix_sigF(1:im,1:jnp,1:mlev)`.

For the sea-level analysis, gridded fields of the quantity $\sigma^{p_{sl}}\sqrt{\rho^{h'}(1)}/2\Omega\bar{\rho}a$ are required in order to calculate the forecast error covariance operator. For the sea-level analysis, this quantity is calculated by calling `dervsigF_slD()` (see Section 7.7.2):

```
call dervsigF_slD(im,jnp,psl_sigF,usl_sigF,vsl_sigF)
```

For the upper-air analysis, gridded fields of $g\sigma^{h}\sqrt{\rho^{hh'}(1)}/2\Omega a$ are calculated by a call to `dervsigF_upD()` (see Section 7.7.3):

```
call dervsigF_upD(im,jnp,mlev,pres_lev,z_sigF,u_sigF,v_sigF)
```

The forecast error standard deviations at the locations of the observations are calculated by interpolation from the arrays `z_sigF`, `u_sigF`, `v_sigF`, `psl_sigF`, `usl_sigF`, `vsl_sigF`, and `mix_sigF`. This is done using a call to `intp_sigF()`:

```
call intp_sigF(im,jnp,mlev,pres_lev,psl_sigF,usl_sigF,        &
               vsl_sigF,z_sigF,u_sigF,v_sigF,mix_sigF,        &
               maxreg,iregbeg, ireglen,ktmax,itypbeg,ityplen, &
               npieces,lat_list,lon_list,pres_list,sigF_list   )
```

The forecast error standard deviations at observation locations are returned in the array `sigF_list(1:npieces)`. The routine `intp_sigF()` is discussed in detail in Section 7.7.4.

4.3.4 Stage III Processing of Covariance Data–Calculation of Analysis Increments

The innovation equation (1) is solved for **x** by calling the routine `solve4x()`:

```
call solve4x(maxreg,iregbeg,ireglen,ityplen,npieces,  &
             kx_list,lat_list,lon_list,pres_list,     &
             sigU_list,sig0_list,sigF_list,1,npieces, &
             dels_list,npieces,xvec                    )
```

The solution **x** is returned in the array `xvec(1:npieces)`. A detailed discussion of `solve4x()` can be found in Section 8, and in [da Silva and Guo, 1996].

The solution **x** to Eqn. (1) is transformed from observation to state space using Eqn. (2), thus producing the AI $\mathbf{w}^a - \mathbf{w}^f$. The AI are returned by the routine `getAinc()`:

```
call getAinc(verbose,stdout,nbandcg,npieces,iregbeg,ireglen,       &
             ityplen,xvec,lat_list,lon_list,pres_list,sigF_list,   &
             im,jnp,mlev,pres_lev,usl_inc,vsl_inc,psl_inc,         &
             u_inc,v_inc,z_inc,mix_inc,usl_sigF,vsl_sigF,psl_sigF, &
             u_sigF,v_sigF,z_sigF,mix_sigF,ktwanted(ktus),         &
             ktwanted(ktvs),ktwanted(ktslp),ktwanted(ktuu),        &
             ktwanted(ktvv),ktwanted(ktHH),ktwanted(ktqq),ier       )
```

The AI are returned on the analysis grid in the following arrays:

- `usl_inc(1:im,1:jnp)`: Sea-level zonal wind u_{sl}.
- `vsl_inc(1:im,1:jnp)`: Sea-level meridional wind v_{sl}.
- `psl_inc(1:im,1:jnp)`: Sea-level pressure p_{sl}.
- `u_inc(1:im,1:jnp,1:mlev)`: Upper-air zonal wind u.
- `v_inc(1:im,1:jnp,1:mlev)`: Upper-air meridional wind v.
- `z_inc(1:im,1:jnp,1:mlev)`: Upper-air geopotential height h.
- `mix_inc(1:im,1:jnp,1:mlev)`: Upper-air water vapor mixing ratio q.

The forecast error standard deviations for the windfield components is calculated on the analysis grid for output. Forecast error standard deviations for the sea-level winds are calculated by calling `dervsigF_slW()`:

```
call dervsigF_slW(im,jnp,psl_sigF,usl_sigF,vsl_sigF)
```

The resulting u and v elements of these standard deviations are returned in the arrays `usl_sigF(1:im,1:jnp)` and `vsl_sigF(1:im,1:jnp)`, respectively.

The upper-air wind forecast error deviations are calculated by calling `dervsigF_upW()`:

```
call dervsigF_upW(im,jnp,mlev,pres_lev,z_sigF,u_sigF,v_sigF)
```

The resulting u and v elements of these standard deviations are returned in `u_sigF(1:im,1:jnp,1:mlev)` and `v_sigF(1:im,1:jnp,1:mlev)`, respectively.

The routine `getAIall()` returns the analysis increments and forecast error standard deviations through its interface. The value returned by `getAIall()` for the analysis error standard deviations are zero.

5 Stage I: Initialization and Processing of Covariance Data

5.1 Overview of the Initialization Routine `initRSRC()`

The resources used by the PSAS are initialized by a call from the analysis interface routines to the routine **`initRSRC()`**. No arguments are passed to **`initRSRC()`**, since **`initRSRC()`** is a driver for other subroutines that initialize the PSAS State I covariance data and other data. The data initialized by **`initRSRC()`** are defined and accessed in include files or modules. The calling tree for **`initRSRC()`** is illustrated in Figure 10.

The routines called by **`initRSRC()`** perform the following tasks:

- **`qtrig0()`**: Initializes look-up table of the sine function values for use by the routine **`qtrig()`**. This table is stored in the **`REAL`** array **`trigtab(1:lentab)`**, defined in the header file **`trigtab.h`**.
- **`ktname0()`**: Fills in the entries of the arrays stored in the common blocks defined in the header file **`kttabl.h`**. Discussed in Section 5.2.
- **`kxname0()`**: Initialize arrays of data source attribute labels, correlation masks, and correlation table indices stored in common blocks defined in the header file **`kxtabl.h`**. Discussed in Section 5.3.
- **`set_OEclas()`**: Determine the set of distinct observation error classes and initialize State I data for observation error standard deviations. Discussed in Section 5.4.
- **`set_OEhCor()`**: Initialize State I tables for observation error horizontal correlation functions. Discussed in Section 5.5.2.
- **`set_OEvCor()`**: Initialize State I tables for observation error vertical correlation coefficients. Discussed in Section 5.5.1.
- **`set_FEhCor()`**: Initialize State I tables for forecast error horizontal correlation functions. Discussed in Section 5.6.2.
- **`set_FEvCor()`**: Initialize State I tables for forecast error vertical correlation coefficients. Discussed in Section 5.6.1.
- **`tabl_FEsigW()`**: Initialize State I data for model functions for streamfunction and velocity potential forecast error standard deviations σ^{ψ} and σ^{χ}, respectively. Discussed in Section 5.7.
- **`tabl_FEalpha()`**: Initialize State I data for model functions for the geostrophic balance parameters α_{um}, α_{ul}, α_{vm}, and α_{vl}. Discussed in Section 5.8.
- **`bands0()`**: Initialize parameters used by the conjugate gradient solver **`conjgr()`**. Discussed in Section 8.
- **`krname0()`**: Initialize the array **`krname`** with the icosahedral region names.
- **`seticos()`**: Set parameters for the icosahedral decomposition.
- **`proxel0()`**: Initialize parameters for the proximity elimination superobbing scheme.
- **`iniAInc()`**: Reset values of **`LOGICAL`** requested AI flags **`ktwanted(1:ktmax)`** based on settings read from the resource file.

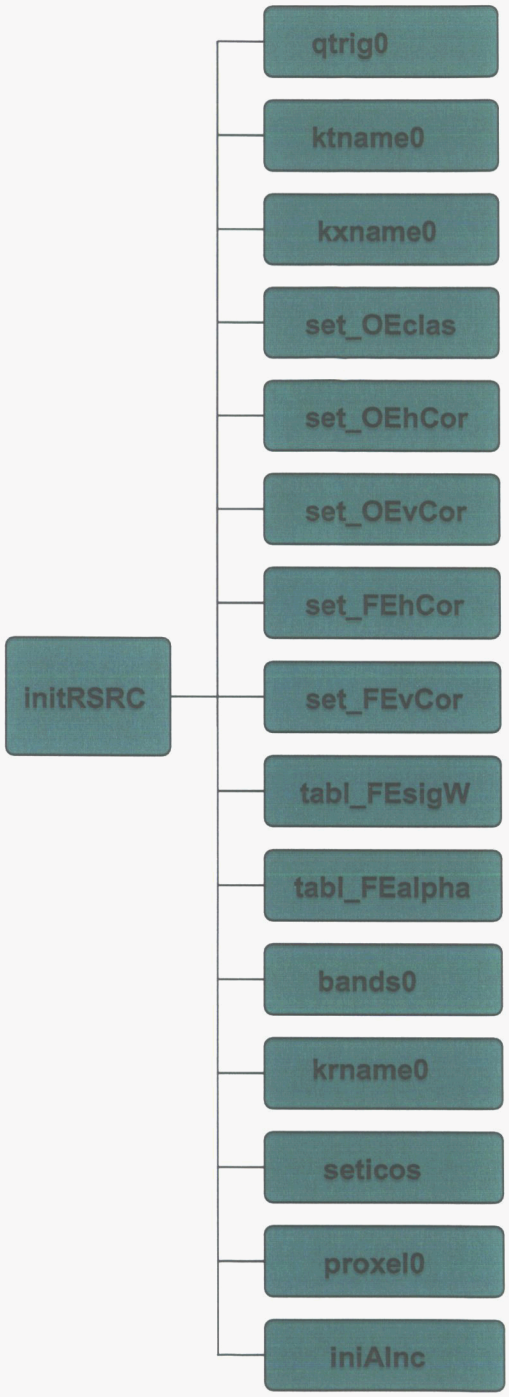


Figure 10: Calling tree for the initialization routine `initRSRC()`. The routines called from `initRSRC()` are listed in the order in which they are called.

5.2 Observation Type and Source Attribute Tables—ktname0()

The routine **ktname0()** (Figure 11) initializes CHARACTER arrays containing the data type name (**ktname**), data type units (**ktunit**), data type description (**ktdesc**), and a LOGICAL mask array (**ktmvar**). The elements of **ktmvar(1:ktmax,1:ktmax)** are set to .TRUE. when pairs of data types are correlated. These arrays are defined in the common blocks **kttabli0** and **kttablc0** defined in the header file **kttabl.h**:

```
common/kttabli0/ktmvar(ktmax,ktmax)
common/kttablc0/ktname(ktmax),ktunit(ktmax),ktdesc(ktmax)
```

This initialization is accomplished by a call to **rdkttbl()**:

```
call rdkttbl(RC_kt,ktmax,ktname,ktunit,ktdesc,ktmvar,istat)
```

The routine **rdkttbl()** which uses the resource label **RC_kt** (defined with value **'DataTypeTable::'** in **kttabl.h**) to locate the this resource:

```
DataTypeTable::

#                                                          ktmvar
#  kt    name       unit        desc                    kt = 1 2 3 4 5 6 7

    1   u_SeaLev   m/sec   Sea Level East-West Wind          $  1
    2   v_SeaLev   m/sec   Sea Level South-North Wind        $  1 1
    3   p_SeaLev   hPa     Sea Level Pressure               $  1 1 1
    4   u_UprAir   m/sec   Upper Air East-West Wind          $  0 0 0 1
    5   v_UprAir   m/sec   Upper Air South-North Wind        $  0 0 0 1 1
    6   H_UprAir   m       Upper Air Geopotential Hight     $  0 0 0 1 1 1
    7   q_UprAir   g/kg    Upper Air Water Vapor Mixing Ratio $  0 0 0 0 0 0 1
```

The first column above is the **kt**-value, the second **ktname(kt)**, the third **ktunit(kt)**, and the fourth **ktdesc(kt)**. The values of **ktmvar** are given at the right. The array **ktmvar** is symmetric, so only the lower-triangular portion of the array is given. If **kt1** and **kt2** are one of $\{1,\ldots,$**ktmax**$\}$, then **ktmvar(kt1,kt2)** $= 1$ whenever the forecast error covariances of **kt1** and **kt2** are both nonzero. In the values of **ktmvar** shown above, error covariances for sea-level pressure and winds are calculated from a multivariate model, error covariances for upper-air geopotential heights and winds from another multivariate model, and upper-air water vapor mixing ratio error covariances are from a univariate model, all in defined in [Guo et al., 1998].

The routine **rdkttbl()** returns an INTEGER status flag **istat**, which is zero unless an error occured.

The resulting arrays are echoed to **stdout** and **ktname0()** returns.

5.3 Observation Data Source Tables—kxname0()

The routine **kxname0()** (Figure 12) initializes the CHARACTER array containing the data source name (**kxclas**), an INTEGER data source rank array (**kxrank**), and a CHARACTER data source description array **kxdesc**. The elements of **kxrank(1:kxmax)** define an order in which data sources are subjected to the superobbing process (see Section 6.4). These arrays are defined in the common blocks **kttablc0** and **kttabli0** defined in the header file **kttabl.h**:

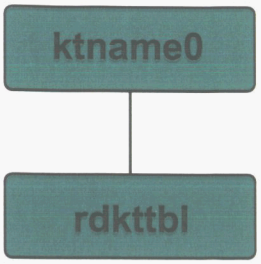


Figure 11: Calling tree for `ktname0()`.

```
common/kxtabli0/kxrank(kxmax),kxtmask(kxmax,ktmax)
common/kxtablc0/kxclas(kxmax),kxdesc(kxmax)
```

The variable `kxtmask()` is initialized by `set_OEclas()` (Section 5.4).

The values of `kxclas`, `kxdesc`, and `kxrank` are set by calling subroutine `rdkxtbl()`:

```
call rdkxtbl(RC_kx,kxmax,kxclas,kxrank,kxdesc,istat)
```

The input `CHARACTER` resource label `RC_kx` (defined with value `'DataSourceTable::'` in the header file `kxtabl.h`) is used to locate the data source table, which contains values of `kxclas`, `kxdesc`, and `kxrank`. A section of this table is shown in Example 5.3.1. The `INTEGER` status flag `istat` is returned with value zero unless an error occurred in `rdkxtbl()`. The resulting arrays are echoed to `stdout` and `kxname0()` returns.

Example 5.3.1: A sample from the section of the resource file read by `kxtbl()` is shown below. The first column contains the `kx` value, the second `observation error class kxclas(kx)` (the type of observation error covariance model used for this data source), the third the value of `kxrank(kx)`, and the fourth a short description of the data source `kxdesc(kx)`. Note that some `kx` values share the same observation error class. For example, the `kx` values 19 through 23 are all cloud track wind measurements, and have the same observation error class (and error covariance model) named `CLDTRKWD`.

```
#  kx    clas      rank      desc
   10  DROPWSND    -500  Dropwinsonde
   11  RADAR_WD    -200  Radar-tracked Rawinsonde
   12  ROCKTSND    -400  Rocketsonde
   13  BALLOON     -800  Balloon
   14  AIR_ASDR   -3000  Aircraft - Air/Sat Relay
   15  AIR_AIDS   -3100  Aircraft - Int. Data Sys
   16  AIR_AIRP   -3200  Aircraft Report
   17  AIR_CODR   -3300  Aircraft Coded Report
   18  AIR_ALPX   -3400  Aircraft - ALPX
   19  CLDTRKWD   -3500  Cld Trk Wind - Wisc E1
   20  CLDTRKWD   -3600  Cld Trk Wind - Wisc E2
   21  CLDTRKWD   -3700  Cld Trk Wind - Wisc W
   22  CLDTRKWD   -3800  Cld Trk Wind - Wisc Ocean
   23  CLDTRKWD   -3900  Cld Trk Wind - Repr. Jap.
   24  CLDTRKWD   -4000  Cld Trk Wind - NESS East
   25  CLDTRKWD   -4100  Cld Trk Wind - NESS West
   26  CLDTRKWD   -4200  Cld Trk Wind - European
   27  CLDTRKWD   -4300  Cld Trk Wind - Japanese
```

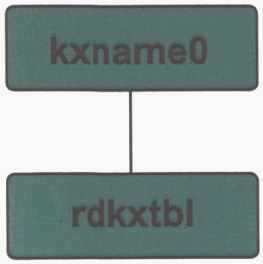


Figure 12: Calling tree for `kxname0()`.

5.4 Determination of Observation Error Classes and State I data for σ_{ou} and σ_{oc}—`set_OEclas()`

The PSAS constructs State I observation error standard deviation tables using a call to the subroutine **`set_OEclas()`** (Figure 13). This routine determines the number of *distinct* observation error classes **`nOEclas`**, assigns an error class index to each data source, and reads appropriate data from the resource file to initialize the State I data for the observation error standard deviations.

Determination of Observation Error Classes: The number of distinct observation error classes is determined by a call to `tabSlist()`:

```
call tabSlist(kxmax,kxclas,kxmax,nOEclas,OEclas)
```

Subroutine **`tabSlist()`** scans the array **`kxclas`**, and returns the number of distinct observation error classes **`nOEclas`**, and a `CHARACTER` array of the error class names `OEclas(1:nOEclas)`, see Example 5.3.1. For **`kx`** $= 19, \ldots, 27$, the observation error class **`kxclas`** is `CLDTRKWD`, and these data sources share the same error covariance model, keyed by the `OEclas` value `CLDTRKWD`.

Once the set of distinct observation error classes has been determined, each data source **`kx`** must be indexed to its observation error class entry in `OEclas`. A call to **`inxSlist()`** returns for each **`kx`** an index `i_kxclas(kx)` to the appropriate observation error class in the array `OEclas`:

```
call inxSlist(nOEclas,OEclas,kxmax,kxclas,i_kxclas)
```

The routine **`inxSlist()`** returns the `INTEGER` index array `i_kxclas(1:kxmax)` that contains a value between `1` and **`nOEclas`** corresponding to the appropriate error class. For the cloud-track winds in Example 5.3.1 the output `i_kxclas` from **`inxSlist()`** is `i_kxclas(19:27)` = `i`, where `OEclas(i)` = `CLDTRKWD`.

The observation error table vertical levels are read in from the resource file by calling the routine **`rdlevels()`**:

```
call rdlevels(RC_OEplev,lvmax_oe,nlev_oe,plev_oe,levtype,iStat)
```

The input `CHARACTER` resource label `RC_OEplev` is a parameter defined in the module `OEclass_tbl` (for v1.5.1, `RC_OEplev` = `'ObsErr*Levels:'`). The input maximum number of

levels `lvmax_oe` is an `INTEGER` parameter defined in the module `config`, where it is set to the parameter `lvmax`, defined in the header file `lvmax.h`. The routine `rdlevels()` returns the number of observation error levels `nlev_oe`, a `REAL` array of their values `plev_oe(1:nlev_oe)`, and a `CHARACTER` level type `levtype`. The `INTEGER` status flag `iStat` is zero unless an error occurred in `rdlevels()`. The section of the resource file read by `rdlevels()` is:

```
ObsErr*Levels:      Pressure  1000 850 700 500 400 300 250 200 150 100 70 50 30 10 5 2 1 0.4
```

Determination of State I Data for σ_{ou} and σ_{oc}: The State I observation error standard deviation tables for σ_{ou} and σ_{oc} are contained in the arrays `sigOu` and `sigOc` respectively (Figure 14). These arrays are defined in the module `OEclass_tbl`, and enter `set_OEclas` through a `USE` statement. Both `sigOu` and `sigOc` are three-dimensional arrays, with the first dimension indexed by vertical level, the second by data type `kt`, and the third by observation error class. For some observation error class and data type values, σ_{ou} and σ_{oc} are defined for only a *subset* of the observation error levels contained in the array `plev_oe`. The first vertical levels for which σ_{ou} and σ_{oc} are defined are specified by `INTEGER` starting level indices `loc_sigOu(1:ktmax,1:nOEclas)` and `loc_sigOc(1:ktmax,1:nOEclas)`, respectively. The number of vertical levels for which σ_{ou} and σ_{oc} are defined is stored in the `INTEGER` arrays `len_sigOu(1:ktmax,1:nOEclas)` and `len_sigOc(1:ktmax,1:nOEclas)`, respectively.

The names of the horizontally correlated and uncorrelated observation error horizontal and vertical correlation functions used for each observation error class are stored in the `CHARACTER` arrays `voecH_u` (ν_{ou}), `voecH_c` (ν_{oc}), and `hoecH_c` (ρ_{oc}).

The initialization of State I data for σ_{ou} and σ_{oc} proceeds as follows:

1. Initialization of the aforementioned observation error standard deviation and correlation variables:

   ```
   voecH_u(1:nOEclas)=' '
   loc_sigOu(1:ktmax,1:nOEclas)=0
   len_sigOu(1:ktmax,1:nOEclas)=0
   sigOu(1:lvmax_oe,1:ktmax,1:nOEclas)=-1.

   hoecH_c(1:nOEclas)=' '
   voecH_c(1:nOEclas)=' '
   loc_sigOc(1:ktmax,1:nOEclas)=0
   len_sigOc(1:ktmax,1:nOEclas)=0
   sigOc(1:lvmax_oe,1:ktmax,1:nOEclas)=-1.
   ```

   The '`-1.`' values assigned to the elements of `sigOu` and `sigOc` signify "undefined."

2. Read in the table values for each observation error class. This is implemented as a loop over the index `iClass` with values between `1` and `nOEclas`. For each value of `iClass`, the following steps are taken:

   (a) Creation of a temporary `CHARACTER` resource label `rc_tmp`:

   ```
   rc_tmp='ObsErr*'//OEclas(iClass)(1:l)//'::'
   ```

   This label will be used to parse the resource file for the appropriate tables of values for the observation error class. There are up to `mxOEt = 2 * ktmax` tables to be read, a table of σ_{ou} and σ_{oc} for each data type `kt` for this observation error class.

   (b) This temporary resource label is used in a call to `rdoetbl()`:

```
call rdoetbl(rc_tmp, mxOEt,n_OEt,name_OEt,   &
             lvmax_oe,lc_OEt,ln_OEt,sig_OEt, &
             ivcHHu,ivcHHc,ihcHHc, iStat      )
```

The routine `rdoetbl()` uses the resource `rc_tmp` to find and read in the `n_OEt` sets of observation error standard deviation tables `sig_OEt` for this class; the starting vertical level `lc_lev(1:n_OEt)` and number of levels `ln_lev(1:n_OEt)`; `INTEGER` flags indicating the type of observation height error correlation function tables `ivcHHu`, `ivcHHc`, and `ihcHHc` for this class. The flag `iStat` is nonzero only if there was an error in `rdoetbl()`.

(c) For each of the `n_OEt` error types the following steps occur:

i. The `CHARACTER` data type name `name_OEt` is trimmed of leading and trailing blanks to its length `l` characters and stored in `eName(1:l)`. The datatype index `kt` is determined by comparing `eName` with a list of datatype names using the function `lstins()`:

```
kt=lstins(ktmax,ktname,eName(1:l),.true.)
```

ii. Nonzero `INTEGER` values of the error correlation function classes `ivcHHu`, `ivcHHc`, and `ihcHHc` are stored in the `CHARACTER` arrays `voecH_u(iClass)`, `voecH_c(iClass)`, and `hoecH_c(iClass)`, respectively (this is done using an internal write).

iii. The `n_OEt` observation error standard deviation types that are used in the error correlation model are stored based on the index `iClass` and the value of `kt` determined by `lstins()`. The σ_{ou} values are stored in `sigOu(:,kt,iClass)`, their `len_sigOu(kt,iClass)` nonzero values beginning with index `loc_sigOu(kt,iClass)`. The σ_{oc} values are stored in `sigOc(:,kt,iClass)`, their `len_sigOc(kt,iClass)` nonzero values beginning with index `loc_sigOc(kt,iClass)`.

(d) For each `kt` $= 1, \ldots,$ `ktmax` the following steps are taken:

i. If `len_sigOu(kt,iClass)` $= 0$, set `loc_sigOu(kt,iClass)` $= 0$

ii. If `len_sigOc(kt,iClass)` $= 0$, set `loc_sigOc(kt,iClass)` $=0$

iii. If either `loc_sigOu(kt,iClass)` > 0 or `loc_sigOc(kt,iClass)` > 0, then the observation error class `iClass` has defined values for either σ_{ou} or σ_{oc}, or both. This is signified by setting the `LOGICAL` flag `KTclas(kt,iClass)` to `.TRUE.`

The final computational step in `set_OEclas()` is to set the elements of the `LOGICAL` array `kxtmask(1:kxmax,1:ktmax)` to `.true.` if `KTclas(kt,i_kxclas(kx)) = .TRUE.`, and `.FALSE.` otherwise.

The resulting observation error standard deviation tables are then echoed to `stdout`.

Example 5.4.1: Consider the case where `kxclas(kx)` in Example 5.3.1 is `CLDTRKWD`, i.e., `kx` is in the set $\{19, \ldots, 27\}$. For this observation error class, the value of `rc_tmp` is `'ObsErr*CLDTRKWD::'`, which refers to the following entry in the resource file:

```
ObsErr*CLDTRKWD::   # kx =  19--27
  u_UprAir.u   4.64  4.62  4.13  3.60  5.80  6.00  6.50  6.50  6.50  7.00  7.00  7.00
7.00  7.00
  v_UprAir.u   4.64  4.62  4.13  3.60  5.80  6.00  6.50  6.50  6.50  7.00  7.00  7.00
7.00  7.00
::
```

The routine `rdoetbl()` interprets this as follows:

- σ_{ou} is defined for u and v, σ_{oc} is not defined.
- Values of upper air zonal wind σ_{ou} for 14 vertical levels follow the label **u_UprAir.u**.
- Values of upper air meridional wind σ_{ou} for 14 vertical levels follow the label **v_UprAir.u**.
- No values for **ivcHHu**, **ivcHHc**, and **ihcHHc** are defined, meaning that observation errors belonging to the class **CLDTRKWD** are uncorrelated.

Example 5.4.2: Consider the observation error class for rawinsonde measurements (**kx** = 7). This class has value of **rc_tmp** = **'ObsErr*RAWINSND::'**, which refers to the following entry in the resource file:

```
ObsErr*RAWINSND::   # kx =   7
  q_UprAir.u   0.76  0.71  0.61  0.24  0.09  0.03
  u_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7
2.7   2.7   2.7   2.7   2.7   2.7
  v_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7
2.7   2.7   2.7   2.7   2.7   2.7
  H_UprAir.u   4.01  5.10  5.66  7.81  9.32 10.59 12.87 14.08 15.88 18.98 21.31 22.81 2
4.29 25.69 27.02 28.28 39.50 30.66
  vCor_HH.u       1
```

The routine **rdoetbl()** interprets this as follows:

- σ_{ou} is defined for q, h, u and v.
- σ_{oc} is not defined.
- Values of upper air zonal wind σ_{ou} for 6 vertical levels follow the label **q_UprAir.u**.
- Values of upper air zonal wind σ_{ou} for 18 vertical levels follow the label **u_UprAir.u**.
- Values of upper air meridional wind σ_{ou} for 18 vertical levels follow the label **v_UprAir.u**.
- Values of upper air geopotential height σ_{ou} for 18 vertical levels follow the label **H_UprAir.u**.
- The value **'1'** is stored in **ivcHHu**, signifying the class horizontally uncorrelated vertical correlation coefficient table used for this observation error class.
- No values for **ivcHHc** and **ihcHHc** are defined, which is consistent with the fact that rawinsonde observation errors are horizontally uncorrelated [Guo et al., 1998, Staff, 1996].

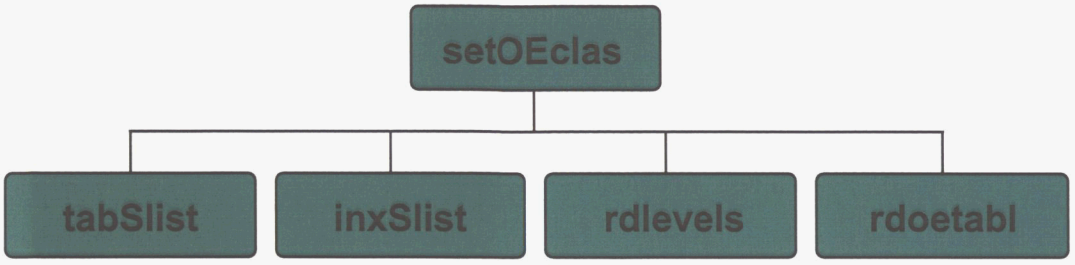


Figure 13: Calling tree for `setOEclas()`.

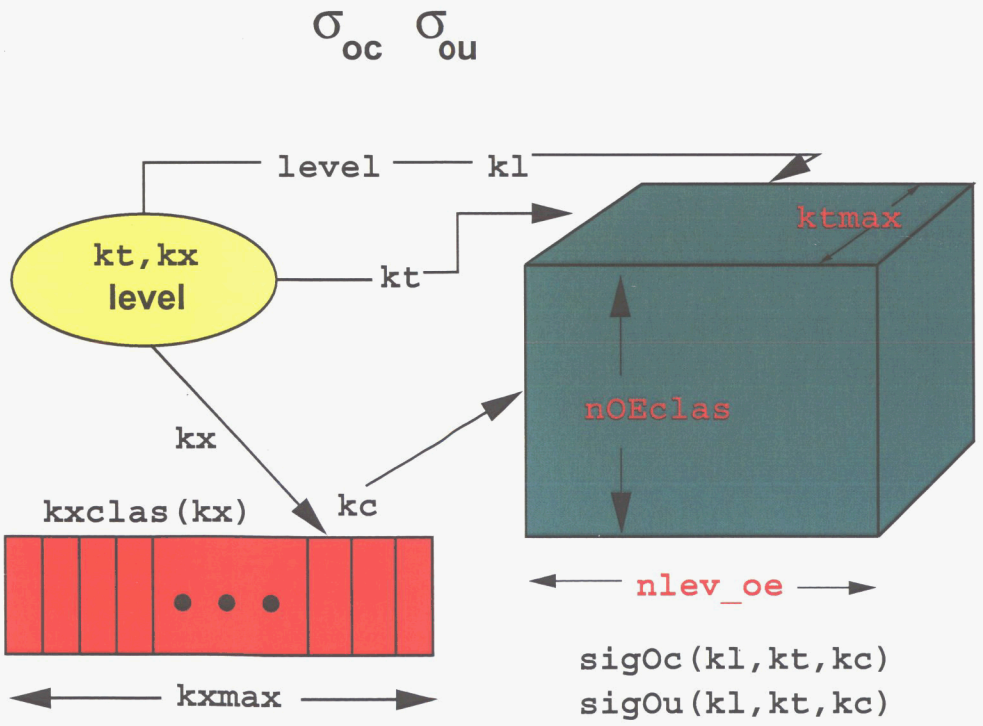


Figure 14: Observation error standard deviation data structures.

5.5 Initialization of State I Observation Error Vertical Correlation Tables

5.5.1 State I Observation Error Vertical Correlations–`set_OEvCor()`

State I data for the vertical correlation coefficients ν_{ou} and ν_{oc} are initialized from the PSAS resource file by the routine **`set_OEvCor()`** (Figure 15). The State I tables for the observation error vertical correlation coefficients ν_{ou} and ν_{oc} are (Figure 16):

- **`n_voecH`** (`INTEGER`): The number of *distinct* vertical correlation tables needed to compute ν_{ou} and ν_{oc} for all the observation error classes determined by the routine **`set_OEclas()`**.
- **`name_voecH`** (`CHARACTER`): An array of names for each vertical correlation coefficient table.
- **`type_voecH`** (`CHARACTER`): An array of vertical correlation coefficient table types.
- **`desc_voecH`** (`CHARACTER`): An array of short descriptions for each vertical correlation coefficient table.
- **`nlev_voecH`** (`INTEGER`): An array containing the number of pressure levels used in each vertical correlation coefficient table.
- **`plev_voecH`** (`REAL`): An array containing the pressure levels used to define each vertical correlation coefficient.
- **`corr_voecH`** (`REAL`): The set of State I vertical correlation coefficient tables for ν_{ou} and ν_{oc}.

These data are defined in the module **`voecH_tbl`** and are accessed by a `USE` module statement.

The routine **`set_OEvCor()`** (Figure 15) determines the number **`n_voecH`** of vertical correlation tables required to compute ν_{ou} and ν_{oc} for each observation error class, creates indices between **`kx`** values and the vertical correlation tables contained in **`corr_voecH`**, and reads from the resource file the values of **`name_voecH`**, **`type_voecH`**, **`desc_voecH`**, **`nlev_voecH`**, **`plev_voecH`**, and **`corr_voecH`**.

Subroutine **`set_OEvCor()`** performs the following steps to initialize the State I data for ν_{ou} and ν_{oc}:

1. Sets the integer variable **`nvc`** to the number of *defined* values in the `CHARACTER` arrays **`voecH_c`** and **`voecH_u`** (**`nvc`** $\leq$ **`2 * nOEclas`**). The defined values of **`voecH_c`** and **`voecH_u`** are stored in the `CHARACTER` array **`kxvoecH(1:nvc)`**.
2. The array **`kxvoecH`** is scanned for redundant entries, implemented in a call to **`tabSlist()`**:

   ```
   call tabSlist(nvc,kxvoecH,MX_voecH,n_voecH,name_voecH)
   ```

   Subroutine **`tabSlist()`** returns the `CHARACTER` array **`name_voecH(1:n_voecH)`** of **`n_voecH`** non-redundant observation error vertical correlation table class names.
3. Create an index between the data source index **`kx`** and the appropriate entry in **`name_voecH`**:
   (a) For values of **`kx`** for which ν_{oc} is defined, assign the appropriate vertical correlation coefficient name:

```
do kx=1,kxmax
  kc=i_kxclas(kx)              ! its OE class index
  kxvoecH(kx)='-'
  if(kc.gt.0) kxvoecH(kx)=voecH_c(kc) ! its voecH_c name
end do
```

(b) Compare the CHARACTER array **kxvoecH** with the non-redundant set of table names in the CHARACTER array **name_voecH** by calling the routine **inxSlist()**:

```
call inxSlist(n_voecH,name_voecH,kxmax,kxvoecH,i_voecHc)
```

The output from **inxSlist()** is the INTEGER index array **i_voecHc(1:kxmax)**. For given **kx** in $\{1,\ldots,$**kxmax**$\}$, **kxvoecH(kx)** = **name_voecH(i_voecHc(kx))**.

4. Steps 3 (a-b) above are repeated for ν_{ou} by storing defined values of **voecH_u** in **kxvoecH**, and calling **inxSlist()**:

```
call inxSlist(n_voecH,name_voecH,kxmax,kxvoecH,i_voecHu)
```

The output from **inxSlist()** is the INTEGER index array **i_voecHu(1:kxmax)**. For given **kx** in $\{1,\ldots,$**kxmax**$\}$, **kxvoecH(kx)** = **name_voecH(i_voecHu(kx))**.

5. The observation error vertical correlation coefficient tables are now read from the resource file. This is implemented as a loop over the index **i=1,n_voecH**. The CHARACTER resource label **rc_tmp** is formed from **name_voecH(i)**, and the table is read by calling **rdvctbl()**:

```
call rdvctbl(rc_tmp,type_voecH(i),desc_voecH(i),     &
             lvmax_vc,nlev_voecH(i),plev_voecH(1,i), &
             corr_voecH(1,1,i),istat                  )
```

Example 5.5.1.1: For rawinsonde data (which have only σ_{ou} defined), the appropriate entry in **name_voecH** is **'-1'**, and the value of **rc_tmp** created by **set_OEvCor()** is **rc_tmp** = **'ObsErr*vCor_HH-1::'**. Below an excerpt is shown from the resource file that is read by **rdvctbl()**:

```
ObsErr*vCor_HH-1::
# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1  .4

  1000   1.
   850   .48  1.
   700   .36  .64  1.
   500   .15  .24  .70  1.
   400   .09  .14  .55  .79  1.
   300   .05  .13  .40  .63  .80  1.
   250   .03  .14  .42  .60  .74  .90  1.
   200   .08  .22  .44  .52  .60  .73  .79  1.
   150   .07  .21  .41  .51  .58  .68  .72  .87  1.
   100   .03  .11  .33  .47  .53  .64  .67  .77  .88  1.
    70   .02  .06  .28  .42  .50  .60  .63  .72  .80  .88  1.
    50   .00  .04  .24  .40  .48  .59  .61  .68  .74  .82  .89  1.
    30  -.01  .02  .04  .05  .07  .16  .21  .44  .56  .65  .72  .78  1.
    10  -.01 -.01  .01  .04  .05  .09  .10  .26  .43  .57  .66  .73  .88  1.
     5   .00 -.03 -.01  .02  .02  .04  .04  .11  .26  .44  .58  .67  .80  .88  1.
     2   .00 -.02 -.02 -.01  .00  .00  .00  .05  .12  .27  .45  .60  .72  .79  .87  1.
     1   .00  .00 -.02 -.02 -.03 -.04 -.05  .00  .06  .13  .28  .49  .65  .71  .77  .86  1.
    .4   .00  .02  .00 -.03 -.04 -.10 -.11 -.07  .00  .06  .15  .33  .56  .64  .69  .74  .84  1.

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1  .4
::
```

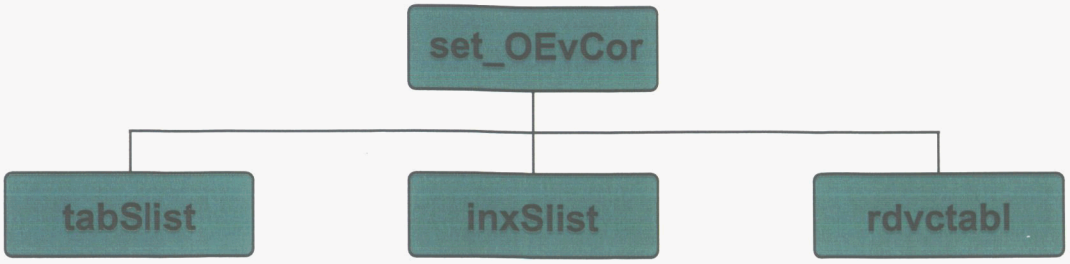


Figure 15: Calling tree for `set_OEvCor()`.

The routine `rdvctbl()` reads this data one line at a time, storing the first entry as a pressure in the array `plev_voecH`, reading the remaining data into `corr_voecH`, and incrementing the value of `nlev_voecH`. Input continues until `rdvctbl()` encounters the delimiter '`::`'[2]. Since no value of `desc_voecH` is defined in the resource file, it is returned blank.

5.5.2 State I Observation Error Horizontal Correlations–`set_OEhCor()`

State I data for the observation error horizontal correlations are initialized from the PSAS resource file by the routine `set_OEhCor()` (Figure 17). The State I tables for the observation error horizontal correlations ρ_{oc} are (Figure 18):

- `n_hoecH` (`INTEGER`): The number of *distinct* observation error horizontal correlation functions tables needed to compute ρ_{oc} for all the observation error classes determined by the routine `set_OEclas()`.
- `name_hoecH` (`CHARACTER`): An array of horizontal correlation function names.
- `type_hoecH` (`CHARACTER`): An array of horizontal correlation types.
- `desc_hoecH` (`CHARACTER`): An array of short descriptions for each horizontal correlation function.
- `nlev_hoecH` (`INTEGER`): An array containing the number of pressure levels on which parameters for the observation error horizontal correlation functions are defined.
- `plev_hoecH` (`REAL`): The pressure levels on which parameters for the observation error horizontal correlation functions are defined.
- `npar_hoecH` (`INTEGER`): The number of parameters for each observation error horizontal correlation function.
- `pars_hoecH` (`REAL`): The set of parameters for the observation error horizontal correlation functions.

The routine `set_OEhCor()` (Figure 17) determines the number `n_hoecH` of observation error horizontal correlation functions for ρ_{oc} for each observation error class, creates indices between `kx` values and the horizontal correlation function parameters contained in `pars_voecH`, and reads from the resource file the values of `name_hoecH`, `type_hoecH`, `desc_hoecH`, `nlev_hoecH`, `plev_hoecH`, `npar_hoecH`, and `pars_hoecH`.

Subroutine `set_OEvCor()` performs the following steps to initialize the State I data for ρ_{oc}:

[2]See the I90 library and its documentation at `ftp://niteroi.gsfc.nasa.gov/pub/packages/i90`

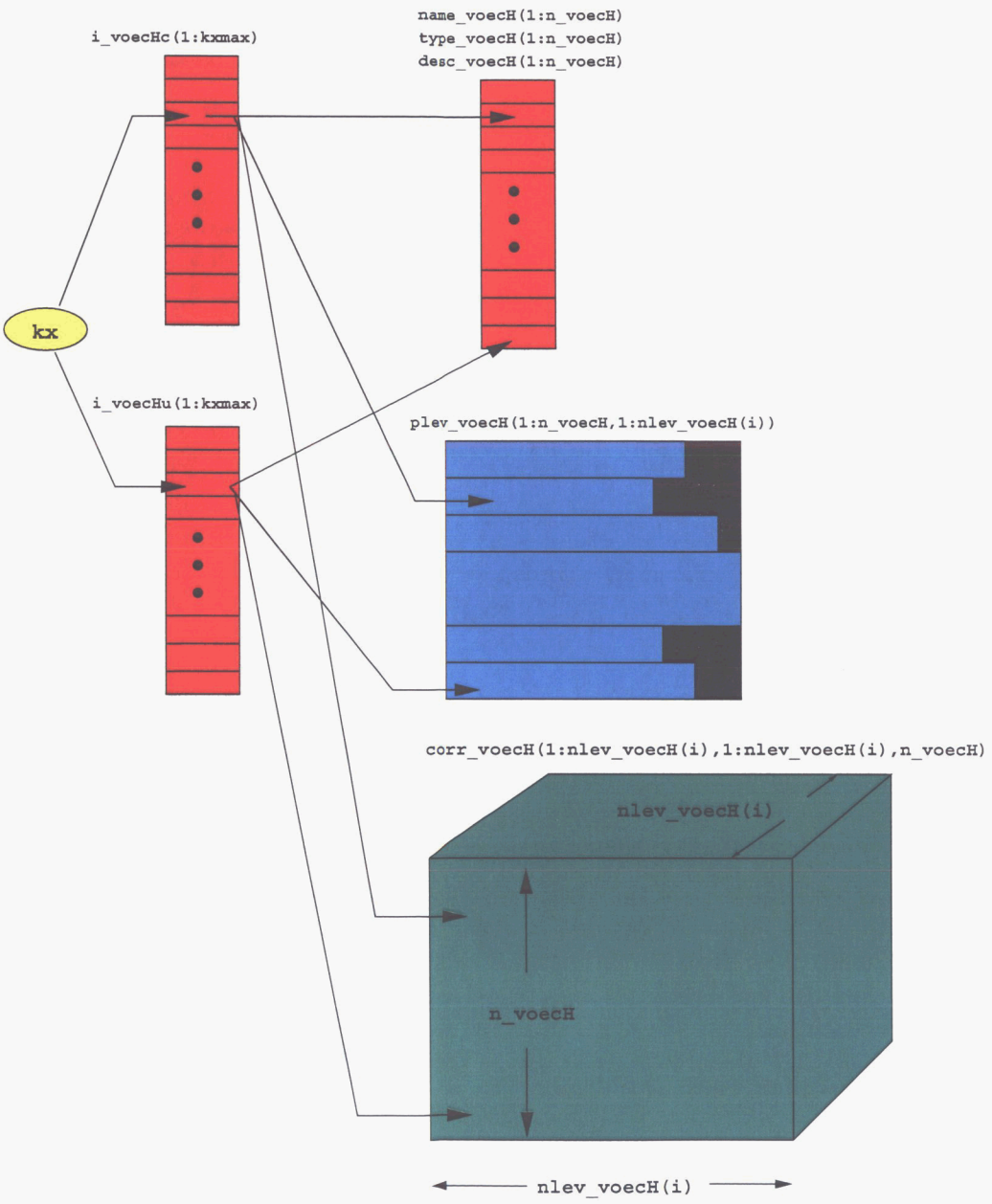


Figure 16: Data structures containing State I tables for ν_{ou} and ν_{oc}.

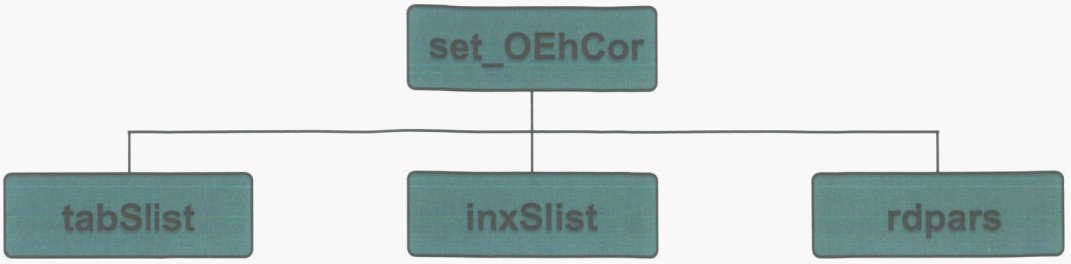


Figure 17: Calling tree for `set_OEhCor()`.

1. Sets the integer variable `nhc` to the number of *defined* values in the `CHARACTER` array `hoecH_c` (`nhc` $\leq$ `nOEclas`). The defined values of `hoecH_c` are stored in the `CHARACTER` array `kxhoecH(1:nhc)`.

2. The array `kxhoecH` is scanned for redundant entries, implemented in a call to `tabSlist()`:

   ```
   call tabSlist(nhc,kxhoecH,MX_hoecH,n_hoecH,name_hoecH)
   ```

   Subroutine `tabSlist()` returns the `CHARACTER` array `name_hoecH(1:n_hoecH)` of `n_voecH` non-redundant observation error horizontal correlation table class names.

3. Create an index between the data source index `kx` and the appropriate entry in `name_hoecH`. This is a two-step process:

   (a) Set the array `kxhoecH(1:kxmax)` so that `kx` values for which ρ_{oc} is defined for the error correlation model are set with the appropriate observation error vertical correlation coefficient name:

   ```
   do kx=1,kxmax
     kc=i_kxclas(kx)                      ! its OE class index
     kxhoecH(kx)='-'
     if(kc.gt.0) kxhoecH(kx)=hoecH_c(kc) ! its hoecH_c name
   end do
   ```

   (b) Compare the `CHARACTER` array `kxhoecH` with the non-redundant set of table names in the `CHARACTER` array `name_hoecH` by calling the routine `inxSlist()`:

   ```
   call inxSlist(n_hoecH,name_hoecH,kxmax,kxhoecH,i_hoecHc)
   ```

   The output from `inxSlist()` is the `INTEGER` index array `i_hoecHc(1:kxmax)`. For given `kx` one of $\{1,\ldots,$ `kxmax`$\}$, `kxhoech(kx)` = `name_hoecH(i_hoecHc(kx))`.

4. The observation error horizontal correlation function parameter tables are now read from the resource file. This is implemented as a loop over the index `i=1,n_hoecH`. The `CHARACTER` resource label `rc_tmp` is formed from `name_hoecH(i)`, and the table is read by calling `rdpars()`:

   ```
   call rdpars(rc_tmp,type_hoecH(i),desc_hoecH(i),          &
               lvmax,nlev_hoecH(i),plev_hoecH(1,i),         &
               MXpar_hc,npar_hoecH(i),pars_hoecH(1,1,i),istat  )
   ```

   The resulting set of data structures is shown in Figure 18.

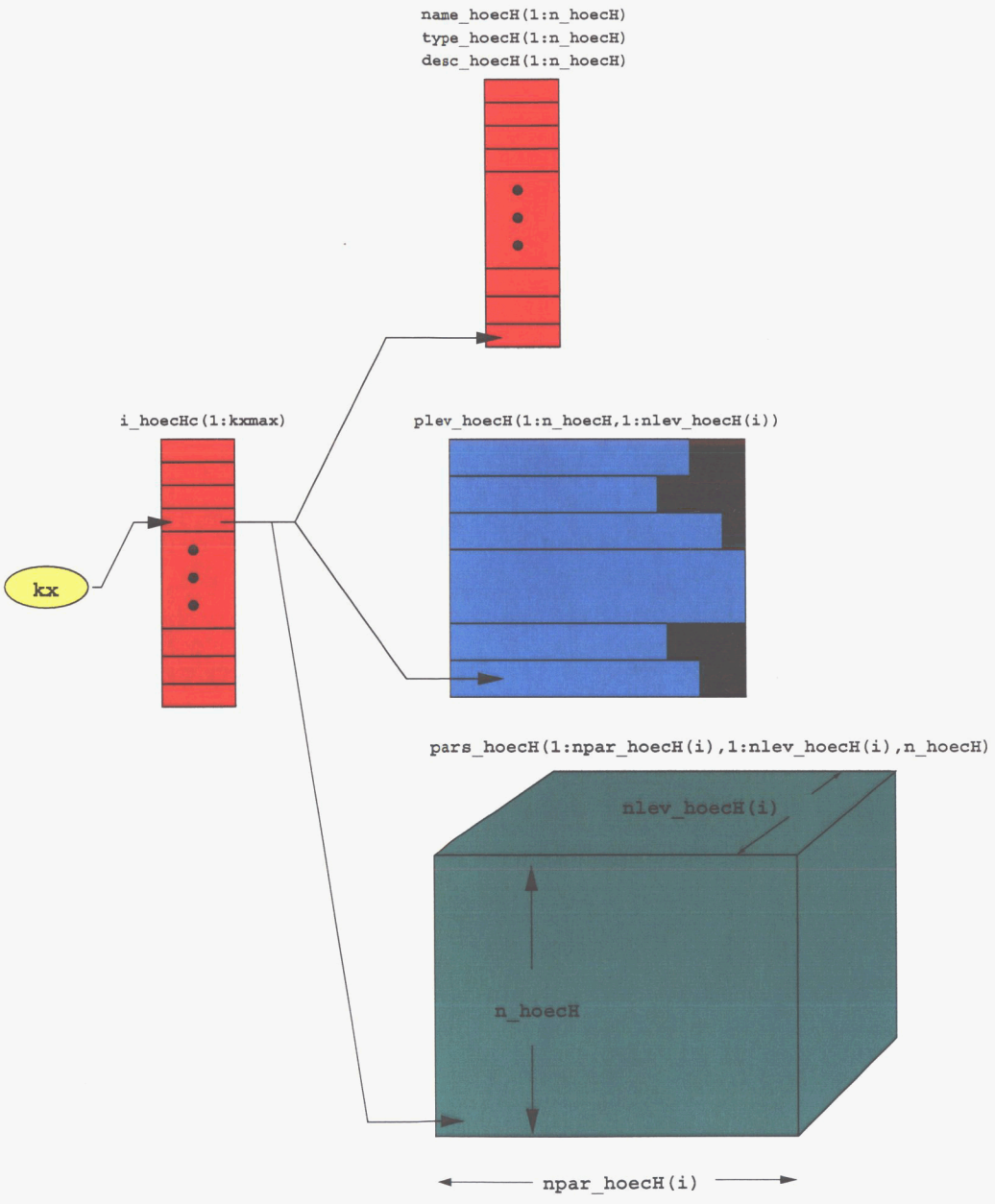


Figure 18: Data structures containing State I Data for ρ_{oc}.

5.6 State I Forecast Error Correlation Data

5.6.1 State I Forecast Error Vertical Correlations—set_FEvCor()

The State I tables for the forecast error vertical correlation coefficients are:

- State I data for upper-air geopotential height and wind forecast errors and sea-level pressure and wind forecast errors (Figure 20):
  - **MX_fecH** (**INTEGER**): The maximum number of forecast error vertical correlation tables. Defined in the header file **MX_hfecH.h**. Currently, **MX_fecH** = 3.
  - **name_vfecH** (**CHARACTER**): An array of names for each vertical correlation coefficient table.
  - **type_vfecH** (**CHARACTER**): An array of types for each vertical correlation coefficient table.
  - **desc_vfecH** (**CHARACTER**): An array of short descriptions for each vertical correlation coefficient table.
  - **nlev_vfecH** (**INTEGER**): An array containing the number of pressure levels used in each State I vertical correlation coefficient table.
  - **plev_vfecH** (**REAL**): An array containing the pressure levels used to define each vertical correlation coefficient.
  - **corr_vfecH** (**REAL**): The set of State I vertical correlation coefficient tables.

  The above data are defined in the module **vfecH_tbl**, and are accessed by a **USE** statement.
- State I data for upper-air water vapor forecast error vertical correlations (Figure 21):
  - **name_vfecQ** (**CHARACTER**): Name of the vertical correlation coefficient table.
  - **type_vfecQ** (**CHARACTER**): The type of vertical correlation coefficient table.
  - **desc_vfecQ** (**CHARACTER**): A short description of the vertical correlation coefficient table.
  - **nlev_vfecQ** (**INTEGER**): The number of pressure levels used in the State I vertical correlation coefficient table.
  - **plev_vfecQ** (**REAL**): An array containing the pressure levels used to define the vertical correlation coefficients.
  - **corr_vfecQ** (**REAL**): The set of State I vertical correlation coefficient tables.

  The above data are defined in the module **vfecQ_tbl**, and are accessed by a **USE** statement.

State I data for the forecast error vertical correlation coefficients are initialized from the PSAS resource file by the routine **set_FEvCor()** (Figure 19).

Subroutine **set_FEvCor()** performs the following steps to initialize the State I forecast error vertical correlation data:

1. Read each of the **MX_fecH** vertical correlation tables from the resource file. This is implemented as a loop over the index **km**, with a call to the routine **rdvctbl()** for each value of **km**:

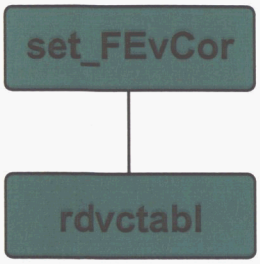


Figure 19: Calling tree for `setFEvCor()`.

```
do km=1,MX_fecH
   call rdvctbl( name_vfecH(km),type_vfecH(km),desc_vfecH(km), &
                 lvmax_vc,nlev_vfecH(km),plev_vfecH(1,km),     &
                 corr_vfecH(1,1,km),istat                       )

end do
```

The vertical correlation model parameters are returned in the array `corr_vfecH`, with the dimensions of the tables of defined entries given by `nlev_vfecH`, and the vertical levels for which the parameters are defined by `plev_vfecH` (Figure 20).

2. The table of vertical correlation model parameters for the forecast upper-air mixing ratio errors are read using a call to `rdvctbl()`:

```
call rdvctbl( name_vfecQ,type_vfecQ,desc_vfecQ,                &
              lvmax_vc,nlev_vfecQ,plev_vfecQ,corr_vfecQ,istat  )
```

The resource label `name_vfecQ` is defined in the module `vfecQ_tbl`. This key is used by `rdvctbl()` to locate and load the State I vertical correlation data. The parameters are returned in `corr_vfecQ(1:nlev_vfecQ,1:nlev_vfecQ)`, and the pressure levels on which the parameters are defined in `plev_vfecQ(1:nlev_vfecQ)`. The name of the vertical correlation model and a short description are stored in the `CHARACTER` variables `type_vfecQ` and `desc_vfecQ`, respectively (Figure 21).

5.6.2 State I Forecast Error Horizontal Correlations—`set_FEhCor()`

The State I data for the forecast error horizontal correlations are:

- State I data for upper-air geopotential height and wind forecast errors and sea-level pressure and wind forecast errors (Figure 23):
  - `MX_fecH` (`INTEGER`): The maximum number of forecast height/wind error correlations. Defined in the include file `MX_hfecH.h`. Currently, `MX_fecH` = 3.
  - `name_hfecH` (`CHARACTER`): An array of names for each forecast height/wind error horizontal correlation function.
  - `type_hfecH` (`CHARACTER`): An array of types for each forecast height/wind error horizontal correlation function.
  - `desc_hfecH` (`CHARACTER`): An array of short descriptions of each forecast height/wind error horizontal correlation function.

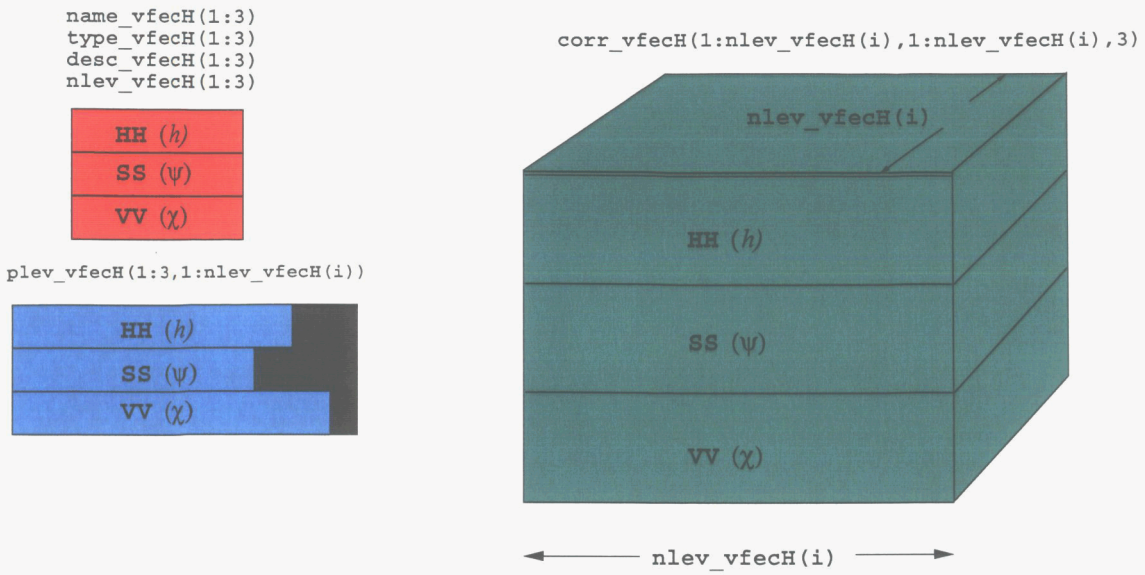


Figure 20: State I data for upper-air height/wind (sea-level pressure/wind) forecast error vertical correlations.

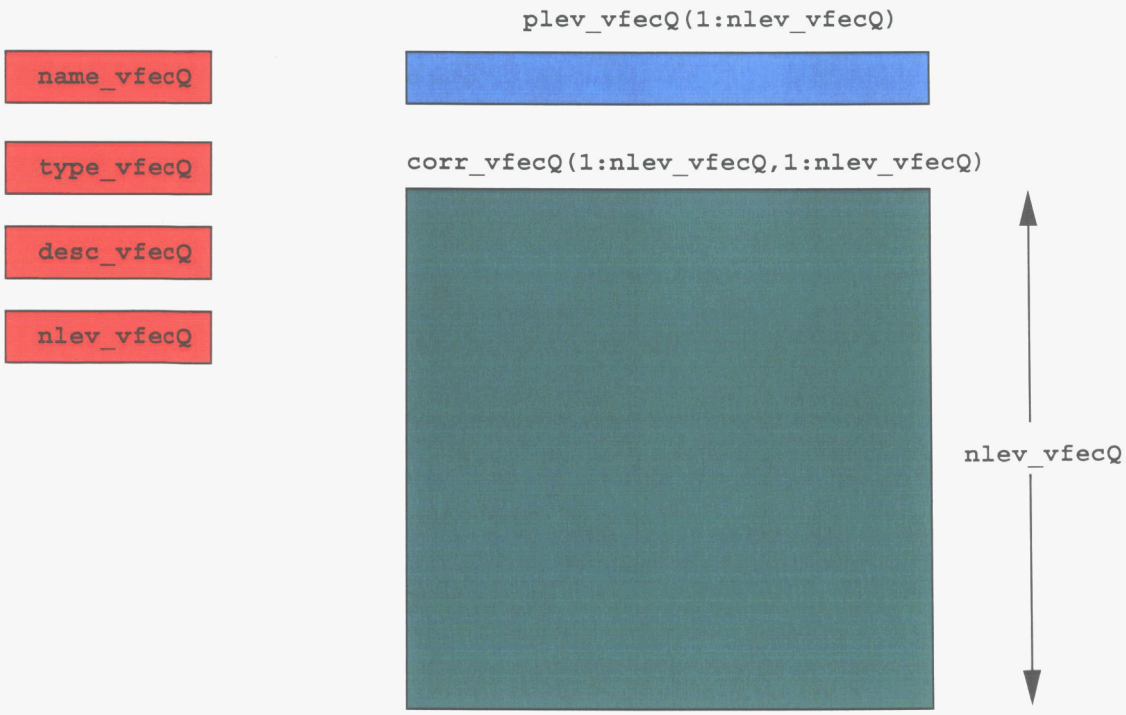


Figure 21: State I data for water vapor mixing ratio forecast error vertical correlations.

- **nlev_hfecH** (**INTEGER**): An array containing the number of pressure levels on which parameters for each forecast height/wind error horizontal correlation function are defined.
- **plev_hfecH** (**REAL**): An array containing the pressure levels on which parameters for each forecast height/wind error horizontal correlation function are defined.
- **pars_hfecH** (**REAL**): The set of parameters used in the height/wind forecast error horizontal correlation functions.

The above data are defined in the module **hfecH_tbl**, and are accessed by a **USE** statement.

- State I data for upper-air water vapor forecast error horizontal correlations (Figure 24):
  - **name_hfecQ** (**CHARACTER**): Name of the forecast water vapor mixing ratio error horizontal correlation function.
  - **type_hfecQ** (**CHARACTER**): The type of forecast water vapor mixing ratio error horizontal correlation function.
  - **desc_hfecQ** (**CHARACTER**): A short description of the forecast water vapor mixing ratio error horizontal correlation function coefficient table.
  - **nlev_hfecQ** (**INTEGER**): The number of pressure levels on which parameters for forecast water vapor mixing ratio error horizontal correlation function are defined.
  - **plev_hfecQ** (**REAL**): An array containing the pressure levels on which parameters for forecast water vapor mixing ratio error horizontal correlation function are defined.
  - **pars_hfecQ** (**REAL**): The set of parameters used in the water vapor mixing ratio forecast error horizontal correlation functions.

  The above data are defined in the module **hfecQ_tbl**, and are accessed by a **USE** statement.

State I data for the forecast error horizontal correlations are initialized from the PSAS resource file by the routine **set_FEhCor()** (Figure 22).

Subroutine **set_FEhCor()** performs the following steps to initialize the State I forecast error horizontal correlation data:

1. The horizontal correlation model parameters are read in by the routine **set_FEhCor()** (Figure 22). The parameters associated with the upper-air wind and height fields (sea-level wind and pressure) are read in the loop shown below. The resource labels **name_hfecH** are defined in the module **hfecH_tbl**

```
do km=1,MX_fecH
   call rdpars(  name_hfecH(km),type_hfecH(km),desc_hfecH(km), &
                 lvmax_hc,nlev_hfecH(km),plev_hfecH(1,km),     &
                 MXpar_hc,npar_hfecH(km),pars_hfecH(1,1,km),   &
                 istat                                          )
end do
```

   The horizontal correlation function parameter tables are returned in the array **pars_hfecH**, with the dimensions of the tables of defined entries given by **npar_hfecH** and **nlev_hfecH**, and the vertical levels for which the parameters are defined by **plev_hfecH** (Figure 23).

2. The horizontal correlation function parameter tables for the forecast upper-air mixing ratio errors are read using a call to **rdpars()**:

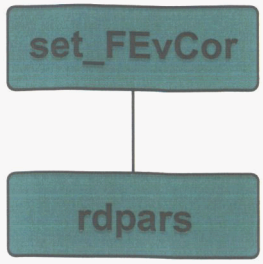


Figure 22: Calling tree for `setFEhCor()`.

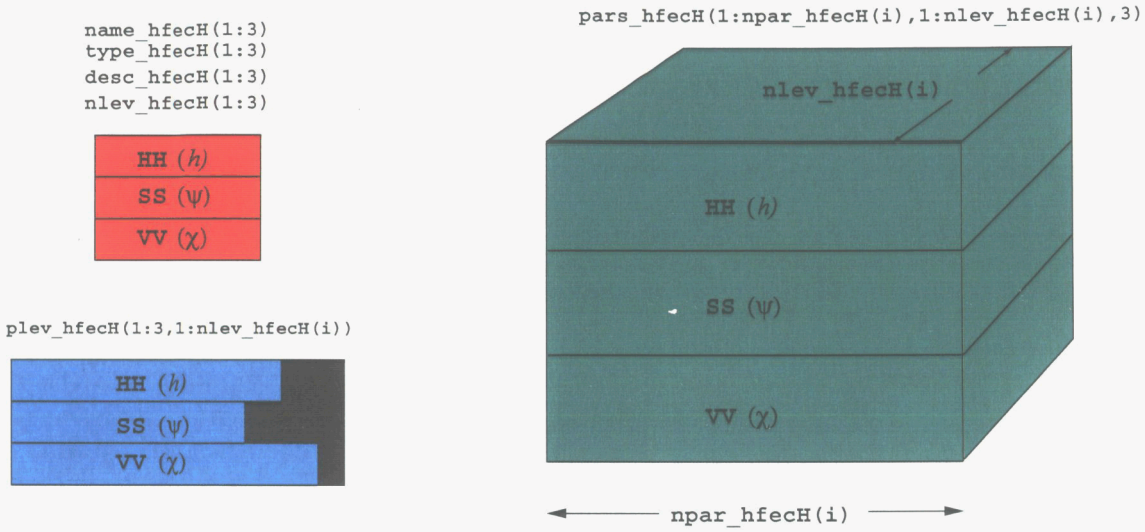


Figure 23: State I data for upper-air height/wind (sea-level pressure/wind) forecast error horizontal correlations.

```
call rdpars( name_hfecQ,type_hfecQ,desc_hfecQ,      &
             lvmax_hc,nlev_hfecQ,plev_hfecQ,        &
             MXpar_hc,npar_hfecQ,pars_hfecQ,istat    )
```

The resource label `name_hfecQ` is defined in the module `hfecQ_tbl`. This variable is used by `rdpars()` to locate and load the model parameter tables. The parameter tables are returned in `pars_hfecQ(1:npar_hfecQ,1:nlev_hfecQ)`, and the pressure levels on which the parameters are defined in `plev_hfecQ(1:nlev_hfecQ)`. The name of the horizontal correlation function and a short description are stored in the `CHARACTER` variables `type_hfecQ` and `desc_hfecQ`, respectively (Figure 24).

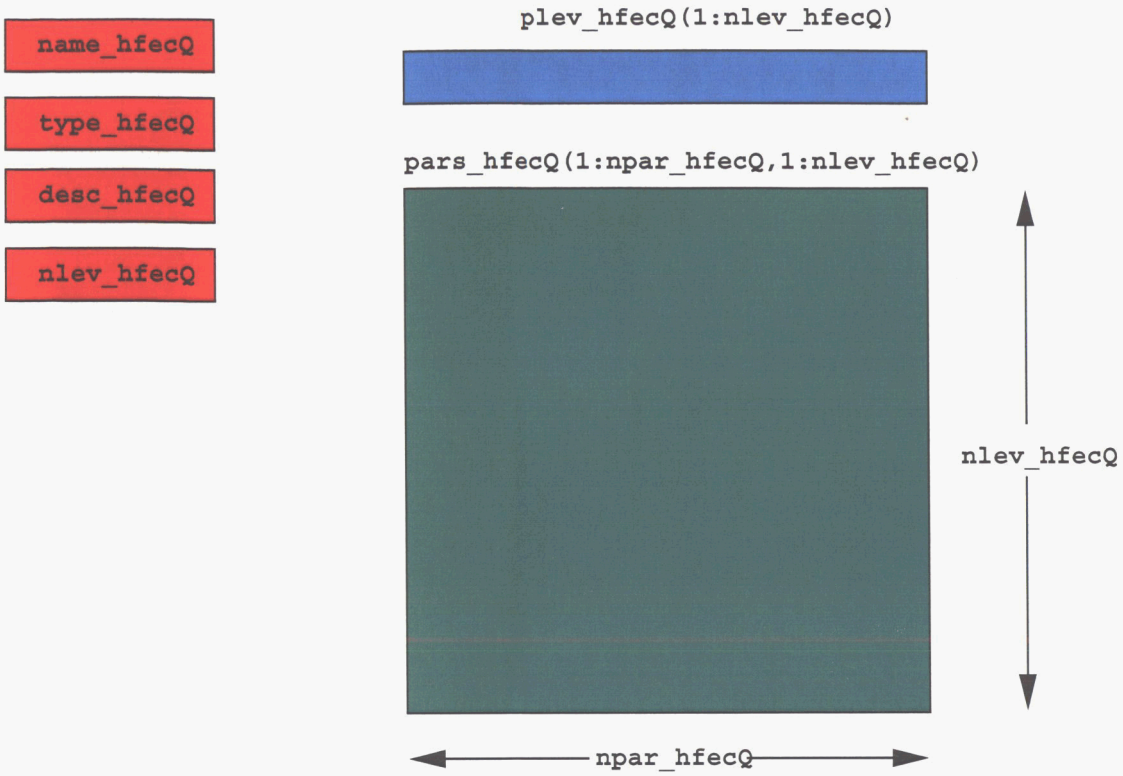


Figure 24: State I data structures mixing ratio forecast error horizontal correlations.

5.7 State I Data for σ^{ψ} and σ^{χ}—tabl_FEsigW()

The State I data for the stream function and velocity potential forecast error standard deviations σ^{ψ} and σ^{χ} are (Figure 26):

- `FEsigW_name` (`CHARACTER`): The name of the functions for σ^{ψ} and σ^{χ}.
- `FEsigW_type` (`CHARACTER`): The type of functions for σ^{ψ} and σ^{χ}.
- `FEsigW_desc` (`CHARACTER`): A short description of the functions for σ^{ψ} and σ^{χ}.
- `FEsigW_nlev` (`INTEGER`): The number of pressure levels on which parameters are defined for the functions for σ^{ψ} and σ^{χ}.
- `FEsigW_plev` (`REAL`): The pressure levels on which parameters are defined for the functions for σ^{ψ} and σ^{χ}.
- `FEsigW_npar` (`INTEGER`): The number of parameters in the functions for σ^{ψ} and σ^{χ}.
- `FEsigW_pars` (`REAL`): The set of parameters used in the functions for σ^{ψ} and σ^{χ}.

The above data are defined in the module `FEsigW_tabl`, and are accessed by a `USE` statement.

The State I data for σ^{ψ} and σ^{χ} are initialized by the routine `tabl_FEsigW()` (Figure 25). This routine issues a call to the routine `rdpars()` to read in the State I data from the resource file:

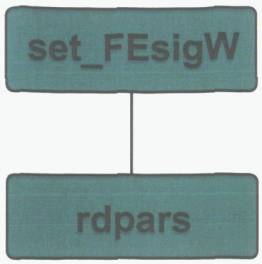


Figure 25: Calling tree for `tabl_FEsigW()`.

```
call rdpars( FEsigW_rsrc,FEsigW_type,FEsigW_desc,          &
             lvmax, FEsigW_nlev,FEsigW_plev,               &
             FEsigW_Mpar, FEsigW_npar,FEsigW_pars, istat    )
```

The input `CHARACTER` resource label `FEsigW_rsrc` is defined (with value `FEsigW_rsrc = 'FcstErr*Sigma_Wind::'`) in the module `FEsigW_tabl`. The routine `rdpars()` parses the resource file and returns the State I data for σ^{ψ} and σ^{χ}. The `INTEGER` flag `istat` is returned with value zero unless an error has occurred in `rdpars()`.

5.8 State I Data for the Geostrophic Balance Parameters α_{um}, α_{ul}, α_{vm}, and α_{vl}—`tabl_FEalpha()`

The State I data for the geostrophic balance parameters α_{um}, α_{ul}, α_{vm}, and α_{vl} are (Figure 28):

- `FEalpha_name` (`CHARACTER`): The name of the functions for α_{um}, α_{ul}, α_{vm}, and α_{vl}.
- `FEalpha_type` (`CHARACTER`): The type of functions for α_{um}, α_{ul}, α_{vm}, and α_{vl}.
- `FEalpha_desc` (`CHARACTER`): A short description of the functions for α_{um}, α_{ul}, α_{vm}, and α_{vl}.
- `FEalpha_nlev` (`INTEGER`): The number of pressure levels on which parameters are defined for the functions for α_{um}, α_{ul}, α_{vm}, and α_{vl}.
- `FEalpha_plev` (`REAL`): The pressure levels on which parameters are defined for the functions for α_{um}, α_{ul}, α_{vm}, and α_{vl}.
- `FEalpha_npar` (`INTEGER`): The number of parameters in the functions for α_{um}, α_{ul}, α_{vm}, and α_{vl}.
- `FEalpha_pars` (`REAL`): The set of parameters used in the functions for α_{um}, α_{ul}, α_{vm}, and α_{vl}.

The above data are defined in the module `FEalpha_tabl`, and are accessed by a `USE` statement.

The State I data for α_{um}, α_{ul}, α_{vm}, and α_{vl} are read from the resource file by the routine `tabl_FEalpha()` (Figure 27). This routine issues a call to the routine `rdpars()`, which inputs the State I data from the resource file:

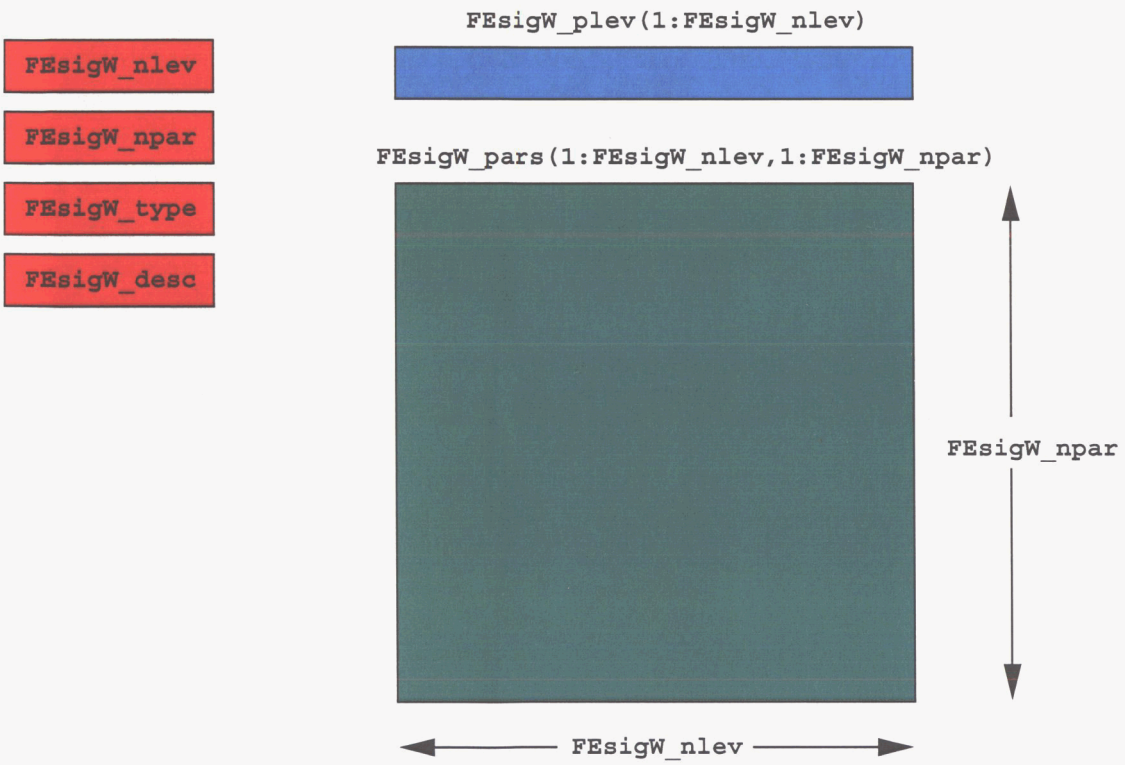


Figure 26: State I data for the streamfunction and velocity potential forecast error standard deviations σ^{ψ} and σ^{χ}.

```
call rdpars(FEalpha_rsrc,FEalpha_type,FEalpha_desc,        &
            lvmax,FEalpha_nlev,FEalpha_plev,               &
            FEalpha_Mpar,FEalpha_npar, FEalpha_pars, istat  )
```

The input resource label `FEalpha_rsrc` passed to `rdpars()` is defined (with value `FEalpha_rsrc = "FcstErr*Aref::"`) in the module `FEalpha_tabl`. The routine `rdpars()` parses this resource and returns the State I data for α_{um}, α_{ul}, α_{vm}, and α_{vl}. The `INTEGER` status flag `istat` returned from `rdpars()` is zero if no error has occurred.

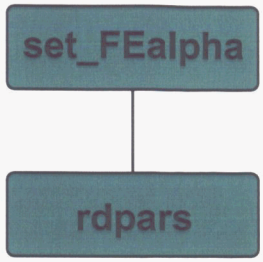


Figure 27: Calling tree for `tabl_FEalpha()`.

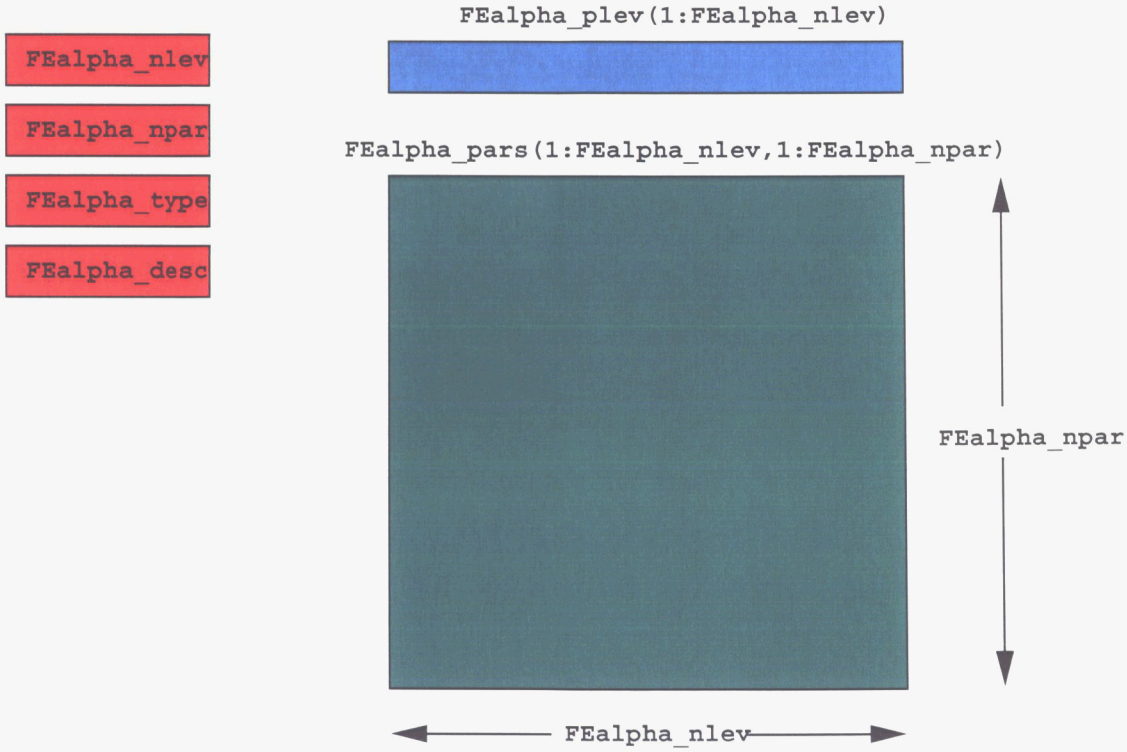


Figure 28: State I data for the geostrophic balance parameters α_{um}, α_{ul}, α_{vm}, and α_{vl}.

6 Processing of Observation Attributes

6.1 Selection of Observations for Analysis—`restrict()`

PSAS uses observational data located within *analysis boxes* defined in the resource file. An analysis box is defined by maximum and minimum:

- Latitude φ.
- Longitude λ.
- Pressure p.
- Time increment from the analysis time Δt.
- Data type index **kt**.
- Data source index **kx**.

Processing of the analysis boxes, and the restriction of observations to those within the analysis boxes is implemented in the subroutine **restrict()** (Figure 29):

```
subroutine restrict (verbose,luverb,nobs,turnoff, &
                     rlats,rlons,rlevs,kx,kt,del, &
                     sigU,sigO,sigF,tstamp,nnobs   )
```

The arguments to **restrict()** are summarized in Table 4.

The routine **restrict()** reads the number of analysis boxes **nboxes** from the resource file. The value of **nboxes** must be less than or equal to the parameter **nbmax**, the maximum number of analysis boxes. The box dimensions are read from the resource file and stored in the array **boxes** (Table 3). Observations with attributes lying outside the analysis boxes are excluded. A **LOGICAL** allocatable array **kl(1:nobs)** is used to record which observations are retained for further processing. If **kl(i) = .TRUE.**, observation attributes with index **i** are included in subsequent processing. After all the observations have been checked for exclusion or inclusion in the analysis, the observation attribute arrays are rearranged to separate the retained and excluded observations.

The routine **setbox()**:

```
call setbox ( nbmax,nboxes,boxes )
```

reads the resource file data selection box settings and returns the number of requested analysis boxes **nboxes**, and a **REAL** array **boxes(2,6,1:nboxes)** of analysis box attributes, listed in Table 3.

The **LOGICAL** array **kl(1:nobs)** of flags is initialized with the data wanted for the current experiment by **kx** and **kt** value through a call to the routine **initkl()**:

```
call initkl ( verbose,luverb,nobs,kx,kt,kl )
```

The set of observations for which **kl** is **.TRUE.** are compared with the analysis boxes defined by the array **boxes** using the routine **llboxes()**:

Table 3: Data box attributes **boxes** returned by **setbox()**.

Element of boxes	Definition
boxes(1,1,1:nboxes)	Minimum **kx** value
boxes(2,1,1:nboxes)	Maximum **kx** value
boxes(1,2,1:nboxes)	Minimum **kt** value
boxes(2,2,1:nboxes)	Maximum **kt** value
boxes(1,3,1:nboxes)	Minimum latitude φ
boxes(2,3,1:nboxes)	Maximum latitude φ
boxes(1,4,1:nboxes)	Minimum longitude λ
boxes(2,4,1:nboxes)	Maximum longitude λ
boxes(1,5,1:nboxes)	Minimum pressure p (hPa)
boxes(2,5,1:nboxes)	Maximum pressure p (hPa)
boxes(1,6,1:nboxes)	Minimum Δt (minutes)
boxes(2,6,1:nboxes)	Maximum Δt (minutes)

```
call llboxes ( verbose,luverb,nboxes,boxes,nobs, &
               rlons,rlats,rlevs,kx,kt,tstamp,kl  )
```

The elements of **kl** corresponding to observations lying outside the dimensions defined in the array **boxes** are marked **.FALSE.**

Finally, the arrays **sigU** (σ_{ou}) and **sigO** (σ_{oc}) are scanned for negative values. This is implemented in a call to **mark_nsig()**:

```
call mark_nsig( nobs,sigU,sigO,kl )
```

If an observation has **kl(i) = .TRUE.**, but has either **sigU(i)** or **sigO(i)** negative, **kl(i)** is set to **.FALSE.**

The attribute arrays are all sorted by the routine **tofront()**:

```
call tofront ( nobs,kx,kt,kl,rlats,rlons,rlevs, &
               del,sigU,sigO,sigF,tstamp,nnobs   )
```

The number of observations with **kl** = **.TRUE.** are counted and returned as **nnobs**. Upon return from **tofront()**, **kl(1:nnobs) = .TRUE.** and **kl(nnobs+1:nobs) = .FALSE.** The other attribute arrays are sorted accordingly.

6.2 Regional Domain Decomposition and Sorting of and Observation Attributes—sort()

Attribute arrays are sorted in the following order as preparation for the block matrix decomposition and for the preconditioning required by the conjugate gradient solver:

- Region index **kr**.

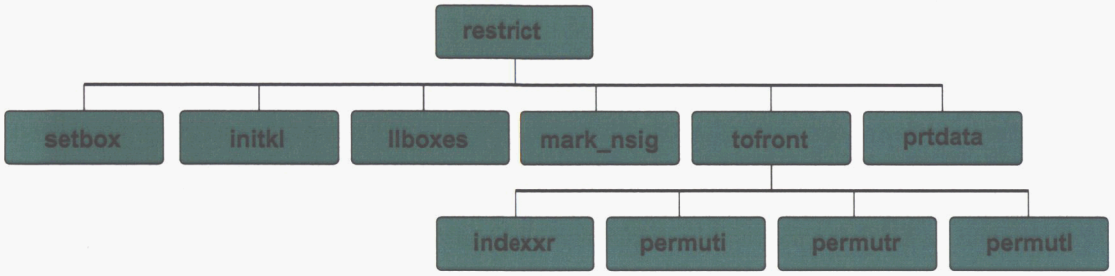


Figure 29: Calling tree for `restrict()`.

Table 4: Data passed into `restrict()` via its interface.

Variable	Type	Intent	Description
verbose	LOGICAL	IN	Verbosity control
luverb	INTEGER	IN	Diagnostic output device
nobs	INTEGER	IN	Input number of observations
rlats	REAL(nobs)	INOUT	Latitude φ
rlons	REAL(nobs)	INOUT	Longitude λ
rlevs	REAL(nobs)	INOUT	Pressure level p
kx	INTEGER(nobs)	INOUT	Data source
kt	INTEGER(nobs)	INOUT	Data type
del	REAL(nobs)	INOUT	$\mathbf{w}^{\mathrm{o}} - H\mathbf{w}^{\mathrm{f}}$
sigU	REAL(nobs)	INOUT	σ_{ou}
sigO	REAL(nobs)	INOUT	σ_{oc}
tstamp	REAL(nobs)	INOUT	Observation time stamp
nnobs	INTEGER	OUT	Input number of observations

- Data type index **kt**.
- Data source index **kx**.
- Latitude φ.
- Longitude λ.
- Vertical level p.

The sorting methodology is explained in Section 3.1.2 (see Figure 5), and is implemented in the subroutine **sort()** (Figure 30):

```
subroutine sort(expid,verbose,luverb,nnobs,rlats,rlons,rlevs, &
                kx,kt,del,sigU,sigO,sigF,tstamp,maxreg,ktmax, &
                iregbeg,ireglen,ityplen                        )
```

The arguments to **sort()** are summarized in Table 5.

In the sort over the index **kr**, the attribute arrays are arranged into **maxreg** regional segments. Each of these regional segments are then sorted according to data type **kt**, data source **kx**, latitude φ, longitude λ, and level p.

The sort over **kr** is implemented in a call to the routine **regsort()**[3]:

```
call regsort( verbose,luverb,nnobs,kx,kt,rlats,rlons,rlevs, &
              del,sigU,sigO,sigF,tstamp,                    &
              maxreg,iregbeg,ireglen,ierr                    )
```

Upon return from **regsort()**, the attribute arrays are arranged into **maxreg** regional segments. The starting indices of the segments are defined in the array **iregbeg(1:maxreg)**, and the length of each segment in **ireglen(1:maxreg)**.

Each regional segment is sorted by data type **kt**, data source **kx**, latitude φ, longitude λ, and level p using the routine **typsort()**:

```
call typsort( verbose,luverb,nnobs,kx,kt,rlats,rlons,rlevs, &
              del,sigU,sigO,sigF,tstamp,                    &
              maxreg,iregbeg,ireglen, ktmax,ityplen          )
```

The number of elements in each **kr**/**kt**-segment of the attribute arrays are returned in the array **ityplen(1:ktmax,1:maxreg)**.

6.3 Elimination of Duplicate Observations—dupelim()

The elimination of duplicate observations is implemented in the routine **dupelim()** (Figure 31):

[3] In the current general implementation of PSAS, the globe is divided into 80 regions using an icosahedral grid. The icosahedral decomposition, and the algorithm used in **regsort()** to assign observations to regions is described in [Pfaendtner, 1996].

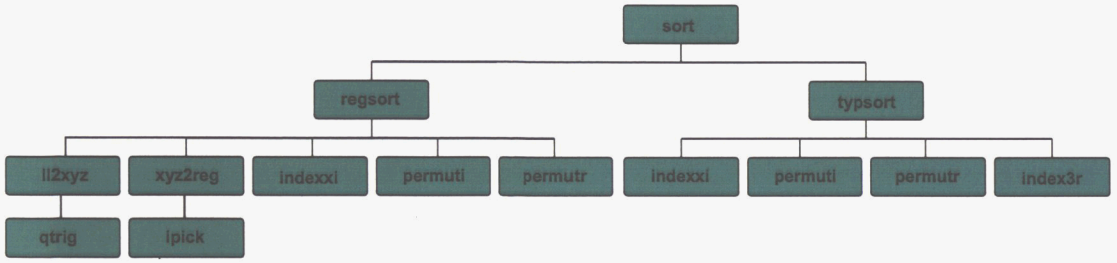


Figure 30: Calling tree for `sort()`.

Table 5: Data passed into `sort()` via its interface.

Variable	Type	Intent	Description
expid	CHARACTER*8	IN	Experiment ID tag
verbose	LOGICAL	IN	Verbosity of diagnostic output
luverb	INTEGER	IN	Diagnostic output device
nnobs	INTEGER	INOUT	Number of observations
rlats	REAL(nnobs)	INOUT	Latitude φ
rlons	REAL(nnobs)	INOUT	Longitude λ
rlevs	REAL(nnobs)	INOUT	Vertical level
kx	INTEGER(nnobs)	INOUT	Data source
kt	INTEGER(nnobs)	INOUT	Data type
del	REAL(nnobs)	INOUT	$\mathbf{w}^{\mathrm{o}} - H\mathbf{w}^{\mathrm{f}}$
sigU	REAL(nnobs)	INOUT	σ_{ou}
sigO	REAL(nnobs)	INOUT	σ_{oc}
sigF	REAL(nnobs)	INOUT	σ_f
tstamp	REAL(nnobs)	INOUT	Observation time stamp
maxreg	INTEGER	IN	Maximum number of regions
ktmax	INTEGER	IN	Maximum number of data types
iregbeg	INTEGER(maxreg)	INOUT	Region start index
ireglen	INTEGER(maxreg)	INOUT	Region length
ityplen	INTEGER(ktmax,maxreg)	INOUT	kr/kt segment lengths

```
subroutine dupelim (verbose,luverb,nnobs,kx,kt,kl,rlats,rlons,rlevs, &
                    del,sigU,sigO,sigF,tstamp,                       &
                    maxreg,  iregbeg, ireglen,ktmax,ityplen           )
```

The arguments to **dupelim()** are summarized in Table 6.

The attribute arrays enter **dupelim()** after they have been sorted by the routine **sort()**. Within each regional segment, the routine **sort()** arranges the observations so that duplicates are in consecutive elements of the arrays **rlat**, **rlon**, and **rlev**. Observations are **not** duplicates if any of the following conditions hold:

- The observations are not colocated:
  - Their pressure level values p differ significantly:

    $$|\texttt{rlev(n)} - \texttt{rlev(n-1)}| > \texttt{TOL}$$

  - Their longitudes λ differ significantly:

    $$|\texttt{rlat(n)} - \texttt{rlat(n-1)}| > \texttt{TOL}$$

  - Their latitudes φ differ significantly:

    $$|\texttt{rlon(n)} - \texttt{rlon(n-1)}| > \texttt{TOL},$$

  where **TOL** is a tolerance, currently, set to 10^{-4}.
- Their data source indices **kx** differ:

  $$\texttt{kx(n)} \neq \texttt{kx(n-1)}$$

- Their data type indices **kt** differ:

  $$\texttt{kt(n)} \neq \texttt{kt(n-1)}$$

If the above test shows that observations **n** and **n-1** are distinct, then **kl(n)** = **.TRUE.**, otherwise **kl(n)** = **.FALSE.** During this elimination process, the region and data type indexing arrays **iregbeg**, **ireglen**, and **ityplen** are updated accordingly.

The attribute arrays are rearranged so that all distinct observations are grouped together at the beginning of the arrays, followed by the duplicates. This separation is done so that the region, data type, data source, and profile sorting hierarchy is retained. This operation is implemented in a call to the routine **tofront()**:

```
call tofront ( nnobs,kx,kt,kl,rlats,rlons,rlevs, &
               del,sigU,sigO,sigF,tstamp,nobs     )
```

The routine **tofront()** counts the number **nobs** of distinct observations. The attribute arrays returned from **tofront()** are arranged so that **kl(1:nobs)** = **.TRUE.** and **kl(nobs+1:nnobs)** = **.FALSE.** The set of attributes with **kl** = **.TRUE.** remain sorted as shown in Figure 5.

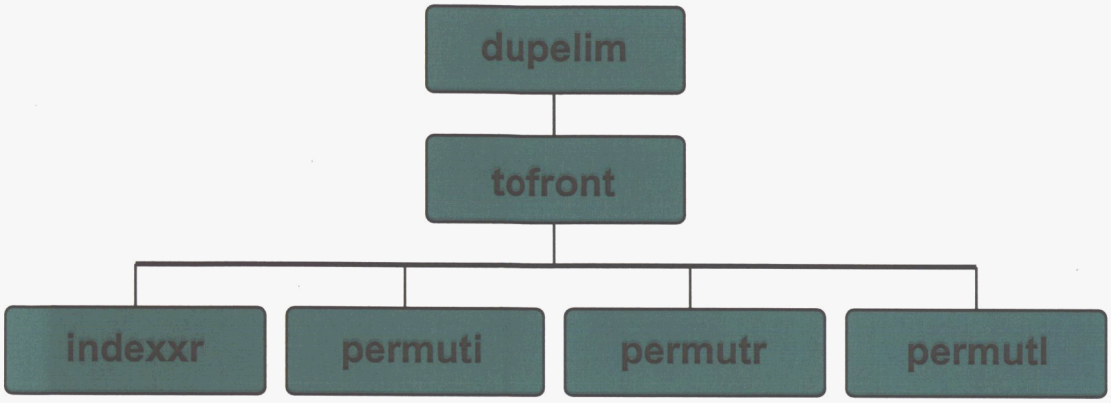


Figure 31: Calling tree for `dupelim()`.

Table 6: Data passed into `dupelim()` via its interface.

Variable	Type	Intent	Description
verbose	LOGICAL	IN	Verbosity of diagnostic output
luverb	INTEGER	IN	Diagnostic output device
nnobs	INTEGER	INOUT	Number of observations
kx	INTEGER(nnobs)	INOUT	Data source
kt	INTEGER(nnobs)	INOUT	Data type
kl	LOGICAL(nnobs)	INOUT	Exclusion flag
rlats	REAL(nnobs)	INOUT	Latitude φ
rlons	REAL(nnobs)	INOUT	Longitude λ
rlevs	REAL(nnobs)	INOUT	Vertical level
del	REAL(nnobs)	INOUT	$\mathbf{w}^{\mathrm{o}} - H\mathbf{w}^{\mathrm{f}}$
sigU	REAL(nnobs)	INOUT	σ_{ou}
sigO	REAL(nnobs)	INOUT	σ_{oc}
sigF	REAL(nnobs)	INOUT	σ_f
tstamp	REAL(nnobs)	INOUT	Observation time stamp
maxreg	INTEGER	IN	Maximum number of regions
iregbeg	INTEGER(maxreg)	INOUT	Region start index
ireglen	INTEGER(maxreg)	INOUT	Region length
ktmax	INTEGER	IN	Maximum number of data types
ityplen	INTEGER(ktmax,maxreg)	INOUT	kr/kt segment lengths

6.4 Superobbing—`proxel()`

The number of observations PSAS uses in an analysis is reduced by consolidating clusters of data with the same data source **kx** and data type **kt** into a single observation. This procedure is commonly referred to as *superobbing*, and is implemented in the routine **proxel()** (Figure 32):

```
subroutine proxel (verbose,luverb,nnobs,kx,kt,kl,rlats, &
                   rlons,rlevs,del,sig_Oc,sig_Ou,sigF,  &
                   tstamp,maxreg,iregbeg,ireglen,       &
                   ktmax1,ityplen,nprox                  )
```

The arguments to **proxel()** are summarized in Table 8.

The input to **proxel()** are observation attribute vectors sorted by region, data type, data source, latitude, longitude, and pressure level. The routine **proxel()** scans the observations sequentially, treating each as a *reference observation* for a scan of all subsequent observations. For each reference observation, **proxel()** checks first whether it has been eliminated by a scan for a previous reference observation. If not, subsequent observations are scanned for inclusion in the *blacklist*. The blacklist is the set of observations targeted for possible elimination through superobbing. A new blacklist is constructed for each reference observation. The observations on the blacklist have the same data type **kt** as the reference observation, and are within chordal distance **Rkm** of the reference observation. The blacklist is complete once all observations subsequent to the reference observation have been scanned for inclusion in the blacklist.

The completed blacklist is scanned by **proxel** for **kx** values to determine the highest ranked data source as defined by the array **kxrank**. The variable **kxrank** is defined in the header file **kxtabl.h** and initialized by the routine **kxname0()**. Observations for the highest-ranked **kx**-value are averaged level-by-level, including observations within log-p distance **dellnp** of each level. The average superob position is calculated in Cartesian coordinates, and subsequently converted to latitude and longitude (φ, λ). Blacklisted observations used to calculate the superob are then eliminated, and the superob attributes are inserted at the appropriate places in the attribute arrays. The region and data type indexing arrays **iregbeg**, **ireglen**, and **ityplen** are updated accordingly.

The main variables used in the superobbing process are:

- Variables defined in the header file **proxel.h**:
  - Horizontal threshold for superobbing **Rkm**.
  - Vertical log-p threshold for superobbing **dellnp**.
  - Angular separation of region centroids to be included in blacklist search **seplim** (parameter).
  - Blacklist data structures listed in Table 7.

  The variables **Rkm** and **dellnp** are initialized from the resource file by the routine **proxel0()**, which was called by **initRSRC()**.
- Allocatable workspace arrays:
  - **LOGICAL** array **tag(nnobs)**, used to record blacklisted observations. For observation **i**, **tag(i)** is **.TRUE.** if the observation is *not* blacklisted, **.FALSE.** if it is blacklisted.

- **REAL** Cartesian coordinate arrays **xobs(nnobs)**, **yobs(nnobs)**, and **zobs(nnobs)**. These variables are used to calculate chordal distances between observations, and for use in determining superob locations.
- **INTEGER** region lengths workspace array **ireglen1(maxreg)**.

The superobbing is initialized by the following steps:

- The Cartesian coordinates on the unit sphere corresponding to each observation's latitude and longitude are calculated through a call to **ll2xyz()**:

```
call ll2xyz ( rlons,rlats,nnobs,xobs,yobs,zobs,ierr )
```

  The Cartesian coordinates x, y, and z are returned in **xobs(1:nnobs)**, **yobs(1:nnobs)**, and **zobs(1:nnobs)**, respectively.

- The number of surviving observations **nobs** is initialized with the input number of observations **nnobs**.
- The number of observations eliminated through superobbing **nprox** is initialized to zero.

Any observation subsequent to the reference observation is added to its blacklist if all of the following criteria are met:

- The chordal distance between the observation and the reference observation is less than **Rkm**.
- The observation has the same data type **kt** as the reference observation.
- The observation has not been previously eliminated **kl = .TRUE.**
- The observation is not already a member of the blacklist; i.e., **tag** is **.TRUE.**

Observations are added to the blacklist by copying their attributes to corresponding blacklist attribute arrays, and incrementing the total number of blacklist observations **ilist**. The reference observation is added to the blacklist when the first subsequent observation has been found suitable for inclusion in the blacklist.

Once a blacklist associated with a given reference observation has been constructed, the highest ranking data source value in the blacklist array **kx_hi** is determined. Data in the blacklist of type **kx_hi** are consolidated in a superob through a call to the routine **prxsob()**:

```
call prxsob ( ierr,kx_hi,did_it,tag,nnobs )
```

The routine **prxsob()** returns a **LOGICAL** flag **did_it** indicating whether or not any observations were eliminated. If **did_it** is **.FALSE.**, no observations were slated for elimination. If **did_it** is **.TRUE.**, observations on the blacklist with **.FALSE.** values in the array **kl_lst** are to be eliminated. The elimination of observations proceeds, and the region index arrays are updated accordingly. The number **nprox** is the number of observations eliminated.

Once the superobbing is complete for all the reference observations, the eliminated observations are separated from the surviving observations by calling **tofront()**:

Table 7: Black list data structures from `proxel.h`

Variable	Type	Description
`nlist`	`INTEGER`	Maximum blacklist size
`idx_lst`	`INTEGER(nlist)`	Index in attribute array
`kx_lst`	`INTEGER(nlist)`	Data source index
`kt_lst`	`INTEGER(nlist)`	Data type index
`ireg_lst`	`INTEGER(nlist)`	Region index
`kl_lst`	`LOGICAL(nlist)`	Elimination flag
`rlats_lst`	`REAL(nlist)`	Latitude φ
`rlons_lst`	`REAL(nlist)`	Longitude λ
`rlevs_lst`	`REAL(nlist)`	Pressure p
`del_lst`	`REAL(nlist)`	$\mathbf{w}^o - H\mathbf{w}^f$
`sigOc_lst`	`REAL(nlist)`	σ_{oc}
`sigOu_lst`	`REAL(nlist)`	σ_{ou}
`sigF_lst`	`REAL(nlist)`	σ_f
`x_lst`	`REAL(nlist)`	Cartesian x on unit sphere
`y_lst`	`REAL(nlist)`	Cartesian y on unit sphere
`z_lst`	`REAL(nlist)`	Cartesian z on unit sphere

```
call tofront ( nnobs,kx,kt,kl,rlats,rlons,rlevs, &
               del,sig_Ou,sig_Oc,sigF,tstamp,nobs )
```

The routine `tofront()` counts the number of surviving observations (for which `kl` = `.TRUE.`), returning the count as `nobs`. The attribute arrays are re-arranged so that `kl(1:nobs)` = `.TRUE.` and `kl(nobs+1:nobs)` = `.FALSE.`. This is done so that the sorting hierarchy shown in Figure 5 is retained. If `nprox` $\neq 0$, the value of `nnobs` is reset to `nobs`.

Upon completion, `proxel()` returns the superobbed observation attribute arrays, adjusted `kr`/`kt`-indexing information contained in `iregbeg`, `ireglen`, and `ityplen`, the number of eliminated observations `nprox`, and the number of surviving observations `nnobs`.

6.5 Separation of Eliminated and Retained Observations—`tofront()`

The routines `restrict()`, `dupelim()` and `proxel()` require that attribute arrays be re-organized to separate surviving data from eliminated data. This operation is implemented in the routine `tofront()`:

```
subroutine tofront (nobs,kx,kt,kl,rlats,rlons,rlevs, &
                    del,sigU,sigO,sigF,tstamp,newnr   )
```

The arguments to `tofront()` are listed in Table 9.

The attribute arrays enter `tofront()` sorted by region, data type and source, and profile. The `LOGICAL` argument `kl` determines which observations are eliminated. Observations are

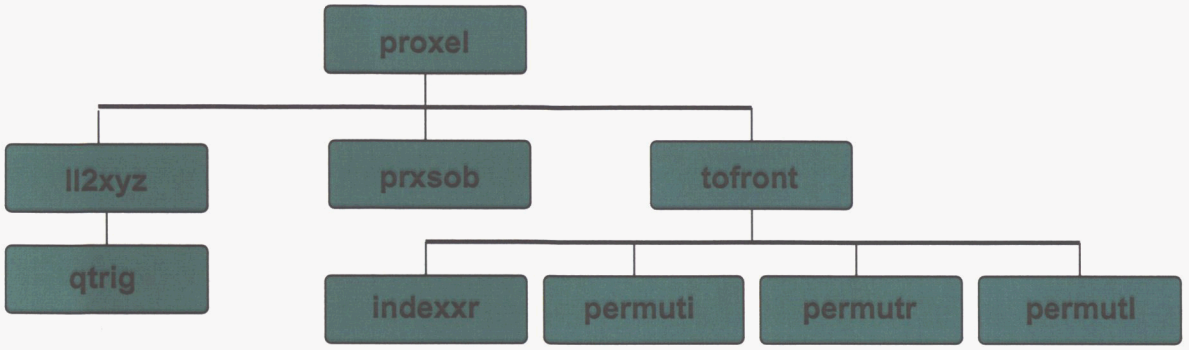


Figure 32: Calling tree for `proxel()`.

Table 8: Data passed into `proxel()` via its interface.

Variable	Type	Intent	Description
verbose	LOGICAL	IN	Verbosity of diagnostic output
luverb	INTEGER	IN	Diagnostic output device
nnobs	INTEGER	INOUT	Number of observations
kx	INTEGER(nnobs)	INOUT	Data source
kt	INTEGER(nnobs)	INOUT	Data type
kl	LOGICAL(nnobs)	INOUT	Exclusion flag
rlats	REAL(nnobs)	INOUT	Latitude φ
rlons	REAL(nnobs)	INOUT	Longitude λ
rlevs	REAL(nnobs)	INOUT	Vertical level
del	REAL(nnobs)	INOUT	$\mathbf{w}^{\mathrm{o}} - H\mathbf{w}^{\mathrm{f}}$
sig_Ou	REAL(nnobs)	INOUT	σ_{ou}
sig_Oc	REAL(nnobs)	INOUT	σ_{oc}
sigF	REAL(nnobs)	INOUT	σ_f
tstamp	REAL(nnobs)	INOUT	Observation time stamp
maxreg	INTEGER	IN	Maximum number of regions
iregbeg	INTEGER(maxreg)	INOUT	Region start index
ireglen	INTEGER(maxreg)	INOUT	Region length
ktmax1	INTEGER	IN	Maximum number of data types
ityplen	INTEGER(ktmax,maxreg)	INOUT	kr/kt segment lengths
nprox	INTEGER	OUT	Number of eliminated observations

retained if `kl = .TRUE.`, and eliminated if `kl = .FALSE.` The number of observations with `kl = .TRUE.` are counted, and this result is stored in the variable `newnr`. The array `kl` and the other observation attribute arrays are rearranged so that `kl(1:newnr) = .TRUE.` and `kl(newnr+1:nobs) = .FALSE.`

The separation process uses a `REAL` array `rsort(1:nobs)` and an `INTEGER` array `iperm(1:nobs)`. Both `rsort` and `iperm` are dynamically allocated.

The elements of `rsort` are calculated using the following procedure:

1. The number of surviving observations `newnr` is initialized to zero.
2. The array `kl(1:nobs)` is scanned for .TRUE. values. If `kl(n) = .TRUE.`, the observation is to be retained in further processing.

```
if( kl(n) ) then
   newnr    = newnr + 1
   rsort(n) = - float(newnr)
else
   rsort(n) = float(n)
endif
```

   The elements of `rsort` are negative for surviving observations, positive for eliminated observations. The negative elements of `rsort` are in reversed order from the sorting hierarchy the observation attributes possess.

3. The ordering of the elements of `rsort` corresponding to `.TRUE.` values of `kl` is reversed to the desired order:

```
flip = - float(newnr)
do n = 1, nobs
   if( kl(n) ) rsort(n) = flip - rsort(n)
continue
```

The array `rsort` is heap-sort indexed by a call to `indexxr()`:

```
call INDEXXR ( nobs,rsort,iperm )
```

The index permutation that will place `rsort(1:newnr)` in the order shown in Figure 5 is returned in the array `iperm`.

The index permutation `iperm` is applied to each of the attribute arrays via calls to `permuti()`, `permutr()`, and `permutl()`:

```
call permuti ( kx,iperm,nobs,kx )
call permuti ( kt,iperm,nobs,kt )

call permutl ( kl,iperm,nobs,kl )

call permutr ( rlats,iperm,nobs,rlats )
call permutr ( rlons,iperm,nobs,rlons )
call permutr ( rlevs,iperm,nobs,rlevs )
call permutr ( del,   iperm, nobs, del   )
call permutr ( sigU,  iperm, nobs, sigU  )
call permutr ( sigO,  iperm, nobs, sigO  )
call permutr ( sigF,  iperm, nobs, sigF  )
call permutr ( tstamp,iperm, nobs, tstamp)
```

Table 9: Data passed into `tofront()` via its interface.

Variable	Type	Intent	Description
`nobs`	`INTEGER`	`INOUT`	Number of observations
`kx`	`INTEGER(nnobs)`	`INOUT`	Data source
`kt`	`INTEGER(nnobs)`	`INOUT`	Data type
`kl`	`LOGICAL(nnobs)`	`INOUT`	Exclusion flag
`rlats`	`REAL(nnobs)`	`INOUT`	Latitude φ
`rlons`	`REAL(nnobs)`	`INOUT`	Longitude λ
`rlevs`	`REAL(nnobs)`	`INOUT`	Vertical level
`del`	`REAL(nnobs)`	`INOUT`	$\mathbf{w}^{\mathrm{o}} - H\mathbf{w}^{\mathrm{f}}$
`sigU`	`REAL(nnobs)`	`INOUT`	σ_{ou}
`sigO`	`REAL(nnobs)`	`INOUT`	σ_{oc}
`sigF`	`REAL(nnobs)`	`INOUT`	σ_f
`tstamp`	`REAL(nnobs)`	`INOUT`	Observation time stamp
`newnr`	`INTEGER`	`OUT`	New number of observations

The final result returned by `tofront()` is the attribute arrays all sorted so that `kl(1:newnr)` = `.TRUE.` and `kl(newnr+1:nobs)` = `.FALSE.`. The two separate segments of the attribute arrays retain the sorting hierarchy shown in Figure 5.

7 Stage II: Processing of Covariance Data

7.1 Interpolation of Observation Error Standard Deviations from State I Tables — `intp_sigO()`

The State II data for the operators Σ_c^o and Σ_u^o are:

- **`sigO_list`** (`REAL`): Horizontally correlated observation error standard deviation Σ_{oc}, evaluated at observation locations.
- **`sigU_list`** (`REAL`): Horizontally uncorrelated observation error standard deviation σ_{ou}, evaluated at observation locations.

The array **`sigO_list`** is defined statically in the analysis interface routine (e.g. in **`getAIall()`**). The array **`sigU_list`** is defined dynamically in the analysis interface routine.

The arrays of σ_{ou} and σ_{oc} are calculated by interpolation from the State I tables **`sigOu`** and **`sigOc`** defined in the module **`OEclass_tbl`** and initialized by **`initRSRC()`** (see Section 5.4).

The interpolation of σ_{ou} and σ_{oc} is performed by the subroutine **`intp_sigO()`** (Figure 33):

```
subroutine intp_sigO(nobs,kxobs,ktobs,rlevs,sigC,sigU)
```

The arguments to **`intp_sigO()`** are summarized in Table 10. The interpolated values of σ_{ou} and σ_{oc} are returned in **`sigO(1:nobs)`** and **`sigU(1:nobs)`**, respectively. The State I observation error standard deviation tables **`sigOc`** and **`sigOu`** enter **`intp_sigO()`** through **`USE`** of the module **`OEclass_tbl`**. The routine **`intp_sigO()`** performs log-linear vertical interpolation from the State I tables to observation level values contained in **`rlevs(1:nobs)`**.

The State I tables of σ_{ou} and σ_{oc} in **`sigOc`** and **`sigOu`** are organized by vertical level, data type **`kt`**, and error class. The routine **`intp_sigO()`** uses the input array **`kxobs`** to determine the observation error class, the array **`ktobs`** to determine the observation data type, and the array **`rlevs`** to determine the observation pressure level. The calculation of σ_{ou} and σ_{oc} from the State I to observation locations is done by log-linear vertical interpolation.

The routine **`intp_sigO()`** performs the following steps:

1. Checks on variables from **`OEclass_tbl`**:
   - The number of levels in the observation error standard deviation tables **`nlev_oe`** > 0.
   - The number of observation error classes **`nOEclas`** > 0.

   If either of these conditions are violated, an error message is written to **`stdout`**, and execution stops.
2. Two temporary work arrays are allocated:
   - **`klev(nobs)`** (`INTEGER`), an index table used in the log-linear vertical interpolation.
   - **`wlev(nobs)`** (`REAL`) , a weight table used in the log-linear vertical interpolation.
3. A call to **`slogtab()`** indexes the observation pressure attribute array **`rlevs(1:nobs)`** against the State I table pressure levels **`plev_oe(1:nlev_oe)`**:

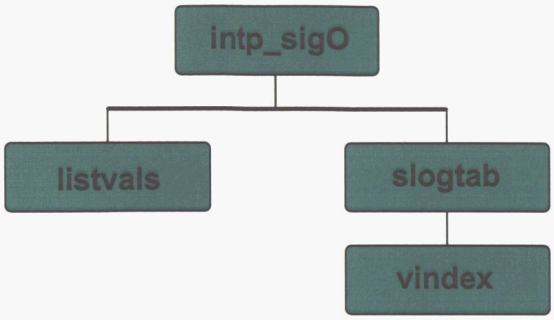


Figure 33: Calling tree for `intp_sigO()`.

```
call slogtab(.not.nearest,nlev_oe,plev_oe,nobs,rlevs,klev,wlev)
```

The indices of the nearest table pressure levels are stored in `klev(1:nobs)`, and the log-linear interpolation weights in `wlev(1:nobs)`.

4. The output arrays `sigO(1:nobs)` and `sigU(1:nobs)` are initialized with the 'undefined' value of -1.

5. For each observation, vertical log-linear interpolation of the values of σ_{oc} and σ_{ou} is performed. Below we outline the the interpolation procedure for σ_{oc} (the steps for σ_{ou} are nearly identical). The index `i` $= 1, \ldots,$ `nobs` refers to an individual observation, for which the following steps are taken:

   (a) Determination of the observation error class `kc` and instrument class `kt`:

   ```
   kc = i_kxclas(kxobs(i))
   kt = ktobs(i)
   ```

   The index array `i_kxclas` is accessed by inclusion of the header file `kxtabl.h`.

   (b) Determination of the starting index `lc`, length `ln`, and ending index `le` of the table to be used for interpolation:

   ```
   lc=loc_sigOc(kt,kc)
   ln=len_sigOc(kt,kc)
   le=lc+ln-1
   ```

   (c) For surface quantities, `sigC` is set directly:

   ```
   sigC(i)=sigOc(lc,kt,kc)
   ```

   (d) For upper-air quantities, the log-linear interpolation is carried out using the table index `klev` and log-linear weight `wlev`:

   ```
   kl=klev(i)
   sigC(i)=sigOc(kl,kt,kc)
   if(kl.lt.le) then
      wt=wlev(i)
      sigC(i)=sigC(i) + wt*(sigOc(kl+1,kt,kc)-sigC(i))
   endif
   ```

Table 10: Data passed into `intp_sigO()` via its interface.

Variable	Type	Intent	Description
nobs	INTEGER	IN	Number of observations
kxobs	INTEGER(nobs)	IN	Data Source
ktobs	INTEGER(nobs)	IN	Data type
rlevs	REAL(nobs)	IN	Vertical level
sigC	REAL(nobs)	OUT	σ_{oc}
sigU	REAL(nobs)	OUT	σ_{ou}

7.2 Determination of the Characteristics of the IMAT Tables

7.2.1 Vertical Level Entries — `merg_plevs()`

All IMAT tables used in the PSAS have at least one vertical dimension. The vertical pressure level values are the same for all the IMATs, and these **nveclev** levels are stored in the vector **pveclev**. Both **nveclev** and **pveclev** are defined in the include file **levtabl.h**. The number of IMAT pressure levels **nveclev**, and their values **pveclev(1:nveclev)** are determined by calls to the subroutine **merg_plevs()** (Figure 34):

```
subroutine merg_plevs(nolev,olevs,nalev,alevs,mxlev,nplev,plevs)
```

The arguments to **merg_plevs()** are summarized in Table 11. The number of IMAT pressure levels **nveclev**, and their values **pveclev** are returned from **merg_plevs()** through the arguments **nplev** and **plevs**, respectively.

The routine **merg_plevs()** merges distinct pressure levels given by the two input lists, **olevs(1:nolevs)** which contains the values of the levels for the input observations, and **alevs(1:nalevs)** which contains the values of the analysis levels, to create a set of distinct pressure levels **plevs(1:nplev)**. The returned list **plevs** is the union of the elements from these two input lists that lie in the range from **pmin** to **pmax**. Only distinct elements within a given precision **pres** from **olevs** and **alevs** are returned in **plevs**.

The routine **merg_plevs()** uses two allocatable work arrays **ilevs(nolev+nalev)** and **lndex(nolev+nalev)**, which are the integral multiple of the minimum step size and a sort permutation array, respectively.

The merge process is a loop that terminates when the number of working levels is less than the maximum number of levels (**nwlev** $<$ **mxlev**), or the working pressure level spacing **rstp** ≤ 0. The maximum and minimum pressure values are given by the **REAL** parameters **pmax** and **pmin**, respectively. For the first iteration, the value of working minimum level resolution **rstp** is determined from the **REAL** parameter **pres** (currently, **pres = 0.01**).

For each iteration of the merge process, the following steps occur:

1. Determination of the range of values for merged level resolution:
   (a) The pressure level truncation step size **smax** is calculated from the current value

of **rstp**:

$$\texttt{smax} = \texttt{rstp} * 10. * * \texttt{floor}(\texttt{log10}(999.)).$$

(b) Calculation of the table resolution step size **smin**, which is based on the minimum pressure level value of **pmin** and working table resolution **rstp**:

$$\texttt{smin} = \texttt{rstp} * 10. * * \texttt{floor}(\texttt{log10}(\texttt{pmin})).$$

2. Initialization of the number of merged levels `l = 0`.

3. The level list **olevs(1:nolevs)** is scanned for valid values that lie between **pmin** and **pmax**. Valid values are truncated using the following scheme:

   (a) For a given element **olevs(i)**, a truncation granularity **sfix** is determined:

   $$\texttt{sfix} = \texttt{min}(\texttt{smax}, \texttt{rstp} * 10. * * \texttt{floor}(\texttt{log10}(\texttt{olevs(i)}))).$$

   (b) The value of **olevs(i)** is rounded to **pfix**, the nearest integral multiple of **sfix**:

   $$\texttt{pfix} = \texttt{sfix} * \texttt{nint}(\texttt{olevs(i)}/\texttt{sfix}).$$

   (c) The rounded pressure value **pfix** is scaled to the current minimum level resolution **smin**. This yields the number **ilevs(l)** of units of **smin** in the rounded pressure **pfix**:

   $$\texttt{ilevs(l)} = \texttt{nint}(\texttt{pfix}/\texttt{smin}).$$

4. The above process is repeated for the level list **alevs(1:nalevs)**, and the current count of the number of valid pressure levels for the merged set is `l`, which is stored in the variable **nwlev**.

5. The set of integer multiples of **smin** contained in **ilev(1:nwlev)** is heap-sort indexed by a call to **indexxi()**:

   ```
   call indexxi(nwlev,ilevs,lndex)
   ```

   The permutation that will put the entries of **ilevs** in ascending order is given by **lndex(1:nwlev)**.

6. The set of levels **ilev(1:nwlev)** is sorted by applying the permutation **lndex(1:nwlev)**, implemented in a call to **permuti()**:

   ```
   call permuti(ilevs,lndex,nwlev,ilevs)
   ```

   The entries of **ilevs** are now in ascending order.

7. The level counter `l` is reset to zero, and the sorted array **ilevs** is scanned to remove redundant entries, with the value of `l` incremented for each *distinct* level value. The number of distinct level values is now given by the counter `l`, whose value is stored in the variable **nwlev**.

8. The number of working levels **nwlev** is compared with the maximum allowable value **mxlev**. If **nwlev** > **mxlev** the working pressure level step size **rstp** must be increased, and the merge process is repeated. The set of values by which **rstp** is increased are (0.01, 0.02, 0.05, 0.1, 0.2, 0.5). If **rstp** = 0.5, and **nwlev** > **mxlev**, **rstp** is then set to -1.0, thus terminating the merge process.

After the merge process, the merged pressure levels are specified by the minimum level spacing **smin** and the array **ilevs** of level values in units of **smin**. The merged pressure levels **plevs** are calculated using the formula below:

```
plevs(1:nplev) = smin * ilevs(1:nplev)
```

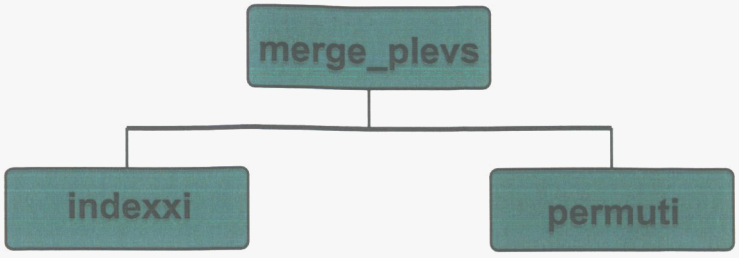


Figure 34: Calling tree for `merg_plevs()`.

Table 11: Data passed into `merg_plevs()` via its interface.

Variable	Type	Intent	Description
`nolev`	`INTEGER`	`IN`	Number of levels in table #1
`olevs`	`REAL(nolev)`	`IN`	Levels in table #1
`nalev`	`INTEGER`	`IN`	Number of levels in table #2
`alevs`	`REAL(nalev)`	`IN`	Levels in table #2
`mxlev`	`INTEGER`	`IN`	Maximum number of levels
`nplev`	`INTEGER`	`OUT`	Number of levels
`plevs`	`REAL(nplev)`	`OUT`	Merged levels

7.2.2 Latitudinal Entries — `merg_lats()`

The IMAT structures for the geostrophic balance parameters α_{um}, α_{ul}, α_{vm}, and α_{vl}, and the mass-decoupled forecast wind error standard deviations σ^{ψ} and σ^{χ} have a latitudinal component. These IMAT structures share a common set of `nveclat` latitudes stored in the array `veclats(1:nveclat)`. The variables `nveclat` and `veclats` are both defined in the module `rlat_imat`, and are accessed through a `USE` module statement.

The IMAT latitude values `veclats` are determined by merging the analysis grid latitude values and the observations' latitude values. This merge is implemented in the routine `merg_lats()` (Figure 35):

```
subroutine merg_lats(ngrdlat,nobslat,obslats, MXveclat,nveclat,veclats)
```

The arguments of `merg_lats()` are summarized in Table 12.

The routine `merg_lats()` uses the input number `ngrdlat` of analysis grid latitudes to calculate grid latitude values, and merges this set with the `nobslat` observation latitudes `obslats(1:nobslat)`. The result is the set of `nveclat` distinct latitudes (`nveclat` $\leq$ `MXveclat`), returned in the array `veclats`.

The merge process comprises the following steps:

1. Allocation of the workspace array of grid latitudes `grdlats(ngrdlat)`.

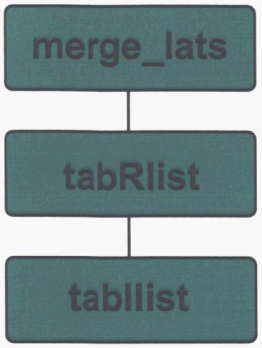


Figure 35: Calling tree for `merg_lats()`.

2. Calculation of the grid point interval `dlat` from the input number of grid latitudes `ngrdlat`:

```
dlat = 180. / (ngrdlat-1)
```

3. Calculation of the grid latitude values:

```
do j=1,ngrdlat
   grdlats(j) = (j-1) * dlat - 90.
end do
```

4. Calculation of an assigned IMAT latitude resolution `res`, which is based on the `INTEGER` argument `MXveclat`:

```
res =180. / (MXveclat - 1)
```

5. The list of grid latitudes `grdlats` is merged into the array `veclats` by a call to the subroutine `tabRlist()`:

```
call tabRlist(res,ngrdlat,grdlats,MXveclat,nveclat,veclats)
```

6. The observation latitudes `obslats(1:nobslat)` are merged into the array `veclats` by a second call to `tabRlist()`:

```
call tabRlist(res,nobslat,obslats,MXveclat,nveclat,veclats)
```

Table 12: Data passed into `merg_lats()` via its interface.

Variable	Type	Intent	Description
`ngrdlat`	`INTEGER`	`IN`	Number of grid latitudes
`nobslat`	`INTEGER`	`IN`	Number of Observation latitudes
`obslats`	`REAL(nobslat)`	`IN`	Observations' latitudes
`MXveclat`	`INTEGER`	`IN`	Maximum number of latitudes
`nveclat`	`INTEGER`	`OUT`	Number of latitudes
`veclats`	`REAL(nveclat)`	`OUT`	Merged latitudes

7.3 Calculation of Forecast Error Correlation IMAT Tables

7.3.1 State II Forecast Height and Wind Error Correlation Tables — `set_fecHH()`

The State II data for the univariate forecast height and wind error correlation functions ρ^h, ρ^ψ, and ρ^χ and their derivatives with respect to the variable $1-\tau$ are:

- Horizontal IMATs and related data defined in the include file `hfecH.h`:

```
real hfecHH,hfecHR,hfecRR,hfecTT
common /hfecHH0/ hfecHH(MXveclev,nHHtab,MX_fecH)
common /hfecHH1/ hfecRR(MXveclev,nHHtab,MX_fecH)
common /hfecHH2/ hfecTT(MXveclev,nHHtab,MX_fecH)
real HHbeg2,Hcoslim,qxHtb1,qxHtb2
common /hfecHH3/        HHbeg2,Hcoslim,qxHtb1,qxHtb2
```

  The IMAT `hfecHH` is the univariate forecast error horizontal correlation function, `hfecRR` and `hfecTT` are its first and second derivatives with respect to $1-\tau$, respectively (Table 13). These IMAT tables are organized by pressure level, the variable $1-\tau$, and function type (i.e., h, ψ, or χ). The variables in the common block `hfecHH3` are described below.

- Vertical correlation IMATs and related data defined in the include file `vfecH.h`:

```
common/vfecHH0/ vfecHH(MXveclev,MXveclev,MX_fecH)
common/vfecHH1/ vfecHD(MXveclev,MXveclev,MX_fecH)
common/vfecHH2/ vfecDD(MXveclev,MXveclev,MX_fecH)
common/vfecHH3/ norm_HH(MXveclev,MX_fecH)
common/vfecHH4/ norm_DD(MXveclev,MX_fecH)
```

  These IMATs are organized by pressure level and function type. The IMAT `vfecHH` is the univariate forecast error normalized vertical correlation coefficients (Table 14), and `norm_HH` and `norm_DD` (Table 15) are normalization factors. The IMAT `norm_HH` is used to normalize the IMATs `vfecHD` and `vfecHH`. The IMAT `norm_DD` is used to normalize `vfecDD` and `vfecHD`.

The IMAT tables for the horizontal and vertical forecast height and wind error correlation functions are constructed by the subroutine `set_fecHH()` (Figure 36). This routine uses

the State I data defined in Sections 5.6.1 and 5.6.2 to calculate the elements of the forecast height/wind error correlation IMATs.

The IMAT tables are input to **set_fecHH()** by inclusion of the include files **hfecHH.h** (horizontal) and **vfecHH.h** (vertical). The State I height and wind data defined in Sections 5.6.1 and 5.6.2 are input by **USE** of the modules **hfecH_tbl** and **hfecH_tbl**, respectively.

The IMAT elements are calculated by interpolation from function values calculated using State I tables initialized by **initRSRC()**. These tables are:

- Horizontal correlation functions:
  - The horizontal correlation function name **name_hfecH**, and type **type_hfecH**.
  - The horizontal correlation function has **npar_hfecH** parameters, stored in the array **pars_hfecH**. These function parameters are level-dependent, and defined on a set of **nlev_hfecH** pressure levels stored in the array **plev_hfecH**.

  These variables are input through **USE** of the module **hfecH_tbl**.
- The vertical correlation coefficients are defined in the array **corr_vfecH**. The number of levels in each vertical correlation coefficient table are stored in the array **nlev_vfecH**, and the level values are stored in **plev_vfecH**. These variables are input through **USE** of the module **hfecH_tbl**.

The univariate forecast error horizontal correlations are functions of the variable $1 - \tau$. There are two ranges of tabulated values of $1 - \tau$ in each horizontal correlation IMAT:

- A fine-scale range of **nHHtb1** values with spacing 1 / **qxHtb1** for $0 \leq 1 - \tau <$ **HHbeg2**.
- A coarse-scale range of **nHHtb2** values with spacing 1 / **qxHtb2** for **HHbeg2** $\leq 1 - \tau < 1 -$ **Hcoslim**.

The inverses **HHinc1** and **HHinc2** are stored in **qxHHtb1** and **qxHHtb2**, respectively.

The routine **set_fecHH()** performs the following steps:

1. The values of **HHbeg2**, **Hcoslim**, **qxHHtb1**, and **qxHHtb2** are calculated using the **REAL** variables **HHmx1**, **HHmx2**, and **dlim**. The steps in this calculation are:
   (a) Calculation of the value of **HHmx1**:

   ```
   HHmx1 = 0.036 * (1. - cos(6030. / rade))
   ```

   where **rade** is the Earth's radius, and enters **set_fecHH()** by a **USE** statement.
   (b) Determination of the value of **dlim**. The maximum value of the correlation cutoff distance value for each class on each vertical level for which function parameters are defined resides in **pars_hfecH**. This set of elements is scanned to determine their maximum value, which is assigned to the variable **dlim**:

   ```
   dlim=0.
   do km=1,MX_fecH
      do lv=1,nlev_hfecH(km)
         if(pars_hfecH(1,lv,km).gt.dlim) dlim=pars_hfecH(1,lv,km)
      end do
   end do
   ```

(c) The value of `dlim` is used to calculate `HHmx2`:

```
HHmx2=1.-cos(dlim/rade)
HHmx2=max(3.*HHmx1,HHmx2)
```

(d) Calculation `HHinc1` and `HHinc2`, the increments of $1-\tau$ for the fine and coarse ranges, respectively:

```
HHinc1=HHmx1/nHHtb1
HHinc2=(HHmx2-HHmx1)/nHHtb2
```

(e) The tabulated values of $1-\tau$ for the IMATs are calculated by calling `intp_ctaus()`:

```
call intp_ctaus(nHHtb1,HHinc1,nHHtb2,HHinc2,nHHtab,ctaus)
```

The tabulated values of $1-\tau$ are returned in `ctaus(1:nHHtb1+nHHtb2)`.

(f) Determination of the variables `HHbeg2` and `Hcoslim` from the array `ctaus`:

```
HHbeg2=ctaus(nHHtb1+1)
Hcoslim=1.-ctaus(nHHtab)
```

(g) Calculation of the values of `qxHtb1` and `qxHtb2`:

```
qxHtb1=1./HHinc1
qxHtb2=1./HHinc2
```

2. The calculation of the IMATs is implemented in a loop over the index `km=1,MX_fecH`, one value for each of the univariate correlations (h, ψ, χ). The following steps occur in the calculation of each correlation function IMAT:

(a) The function type `type_hfecH(km)` is checked to ensure it is defined. If not, an error message is written to `stderr` and execution stops.

(b) Tabulated values of the forecast error horizontal correlation function and its first and second derivatives with respect to $1-\tau$ are calculated by calling `intp_hCor()`:

```
call intp_hCor( name_hfecH(km), type_hfecH(km),                &
                nlev_hfecH(km),plev_hfecH(1,km),               &
                MXpar_hc,  npar_hfecH(km),pars_hfecH(1,1,km), &
                nveclev,pveclev, nHHtab, ctaus,                &
                2,MXveclev,hfecHH(1,1,km),                     &
                hfecRR(1,1,km),hfecTT(1,1,km)                   ) .
```

The horizontal correlation function IMAT is returned in `hfecHH(1:nHHtab,1:nveclev,km)`, its first derivative with respect to $1-\tau$ in `hfecRR(1:nHHtab,1:nveclev,km)`, and its second derivative with respect to $1-\tau$ in `hfecTT(1:nHHtab,1:nveclev,km)`.

(c) The IMATs representing horizontal correlation functions index non-separable forecast error correlation functions described in [Gaspari and Cohn, 1999]. The IMATs calculated above are normalized by a factor .5 since cross-correlation functions in the horizontal are formed by averaging horizontal autocorrelation functions (see [Gaspari et al., 1998] Section 3):

```
hfecHH(1:nveclev,1:nHHtab,km)=.5*hfecHH(1:nveclev,1:nHHtab,km)
hfecRR(1:nveclev,1:nHHtab,km)=.5*hfecRR(1:nveclev,1:nHHtab,km)
hfecTT(1:nveclev,1:nHHtab,km)=.5*hfecTT(1:nveclev,1:nHHtab,km)
```

(d) The vertical correlation IMAT table `vfecHH` is calculated by a call to `intp_vCor()`:

```
call intp_vCor( name_vfecH(km),                              &
                lvmax_vc,nlev_vfecH(km),plev_vfecH(1,km), &
                corr_vfecH(1,1,km),                          &
                MXveclev,nveclev,pveclev, vfecHH(1,1,km)   )
```

The vertical correlation function values for this class are returned in the array `vfecHH(1:nveclev,1:nveclev,km)`.

Table 13: Forecast Height/Wind Error Horizontal Correlation IMATs

IMAT Structure	Quantity
`hfecHH(1:nveclev,1:nHHtab,1)`	$\rho^{h}(\tau_{ij};p_i)$
`hfecHH(1:nveclev,1:nHHtab,2)`	$\rho^{\psi}(\tau_{ij};p_i)$
`hfecHH(1:nveclev,1:nHHtab,3)`	$\rho^{\chi}(\tau_{ij};p_i)$
`hfecRR(1:nveclev,1:nHHtab,1)`	$\rho^{h'}(\tau_{ij};p_i)$
`hfecRR(1:nveclev,1:nHHtab,2)`	$\rho^{\psi'}(\tau_{ij};p_i)$
`hfecRR(1:nveclev,1:nHHtab,3)`	$\rho^{\chi'}(\tau_{ij};p_i)$
`hfecTT(1:nveclev,1:nHHtab,1)`	$\rho^{h''}(\tau_{ij};p_i)$
`hfecTT(1:nveclev,1:nHHtab,2)`	$\rho^{\psi''}(\tau_{ij};p_i)$
`hfecTT(1:nveclev,1:nHHtab,3)`	$\rho^{\chi''}(\tau_{ij};p_i)$

(e) Calculation of the normalization factors `norm_HH` and `norm_DD`:

```
do k=1,nveclev
  norm_HH(k,km) = 1./sqrt( vfecHH(k,k,km)*2.*hfecHH(k,1,km) )
  norm_DD(k,km) = 1./sqrt( vfecHH(k,k,km)*2.*hfecRR(k,1,km) )
end do
```

(f) The normalization factors `norm_HH` and `norm_DD` are used to compute the normalized `vfecHD` and `vfecDD`:

```
do l=1,nveclev
  do k=1,nveclev
    vfecHD(k,l,km) = vfecHH(k,l,km) * norm_HH(k,km) * norm_DD(l,km)
    vfecDD(k,l,km) = vfecHH(k,l,km) * norm_DD(k,km) * norm_DD(l,km)
  end do
end do
```

(g) Normalization of `vfecHH`:

```
do l=1,nveclev
  do k=1,nveclev
    vfecHH(k,l,km) = vfecHH(k,l,km) * norm_HH(k,km) * norm_HH(l,km)
  end do
end do
```

7.3.2 State II Forecast Water Vapor Mixing Ratio Error Correlation Tables — `set_fecQQ()`

The elements of the operator C^q associated with forecast water-vapor mixing ratio error correlations are calculated using tabulated values of the forecast water-vapor mixing ratio error correlation function ρ^q. The State II data for ρ^q are:

- The forecast water vapor mixing ratio horizontal error correlation function tables and related data defined in the include file `hfecQQ.h`:

```
common /hfecQQ0/ hfecQQ(MXveclev,nQQtab)
common /hfecQQ1/ QQbeg2,Qcoslim,qxQtb1,qxQtb2
```

Table 14: Forecast Height/Wind Error Vertical Correlation IMATs

IMAT Structure	Quantity
`vfecHH(1:nveclev,1:nveclev,1)`	$\frac{\nu^h(p_i,p_j)}{2[\nu^h(p_i,p_i)\rho^h(1;p_i)\nu^h(p_j,p_j)\rho^h(1;p_j)]^{1/2}}$
`vfecHH(1:nveclev,1:nveclev,2)`	$\frac{\nu^\psi(p_i,p_j)}{2[\nu^\psi(p_i,p_i)\rho^\psi(1;p_i)\nu^\psi(p_j,p_j)\rho^\psi(1;p_j)]^{1/2}}$
`vfecHH(1:nveclev,1:nveclev,3)`	$\frac{\nu^\chi(p_i,p_j)}{2[\nu^\chi(p_i,p_i)\rho^\chi(1;p_i)\nu^\chi(p_j,p_j)\rho^\chi(1;p_j)]^{1/2}}$
`vfecHD(1:nveclev,1:nveclev,1)`	$\frac{\nu^h(p_i,p_j)}{2[\nu^h(p_i,p_i)\rho^h(1;p_i)\nu^h(p_j,p_j)\rho^{h\prime}(1;p_j)]^{1/2}}$
`vfecHD(1:nveclev,1:nveclev,2)`	$\frac{\nu^\psi(p_i,p_j)}{2[\nu^\psi(p_i,p_i)\rho^\psi(1;p_i)\nu^\psi(p_j,p_j)\rho^{\psi\prime}(1;p_j)]^{1/2}}$
`vfecHD(1:nveclev,1:nveclev,3)`	$\frac{\nu^\chi(p_i,p_j)}{2[\nu^\chi(p_i,p_i)\rho^\chi(1;p_i)\nu^\chi(p_j,p_j)\rho^{\chi\prime}(1;p_j)]^{1/2}}$
`vfecDD(1:nveclev,1:nveclev,1)`	$\frac{\nu^h(p_i,p_j)}{2[\nu^h(p_i,p_i)\rho^{h\prime}(1;p_i)\nu^h(p_j,p_j)\rho^{h\prime}(1;p_j)]^{1/2}}$
`vfecDD(1:nveclev,1:nveclev,2)`	$\frac{\nu^\psi(p_i,p_j)}{2[\nu^\psi(p_i,p_i)\rho^{\psi\prime}(1;p_i)\nu^\psi(p_j,p_j)\rho^{\psi\prime}(1;p_j)]^{1/2}}$
`vfecDD(1:nveclev,1:nveclev,3)`	$\frac{\nu^\chi(p_i,p_j)}{2[\nu^\chi(p_i,p_i)\rho^{\chi\prime}(1;p_i)\nu^\chi(p_j,p_j)\rho^{\chi\prime}(1;p_j)]^{1/2}}$

Table 15: Forecast Height/Wind Error Correlation Normalization Factors

IMAT Structure	Quantity
`norm_HH(1:nveclev,1)`	$[\nu^h(p_i,p_i)\rho^h(1;p_i)]^{-1/2}$
`norm_HH(1:nveclev,2)`	$[\nu^\psi(p_i,p_i)\rho^\psi(1;p_i)]^{-1/2}$
`norm_HH(1:nveclev,3)`	$[\nu^\chi(p_i,p_i)\rho^\chi(1;p_i)]^{-1/2}$
`norm_DD(1:nveclev,1)`	$[\nu^h(p_i,p_i)\rho^h(1;p_i)]^{-1/2}$
`norm_DD(1:nveclev,2)`	$[\nu^\psi(p_i,p_i)\rho^{\psi'}(1;p_i)]^{-1/2}$
`norm_DD(1:nveclev,3)`	$[\nu^\chi(p_i,p_i)\rho^{\chi'}(1;p_i)]^{-1/2}$

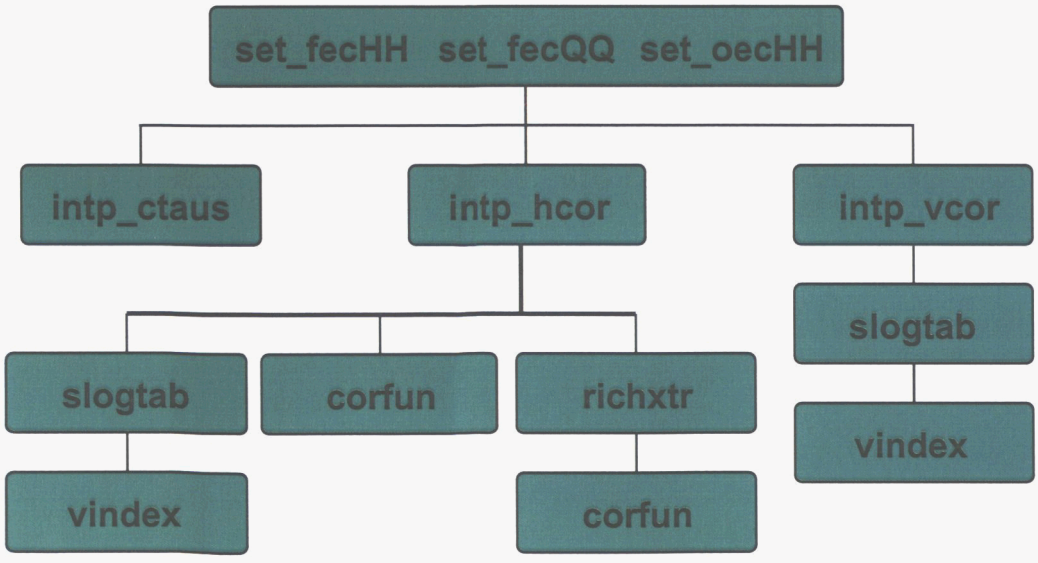


Figure 36: Calling tree for `set_fecHH()`, `set_fecQQ()`, and `set_oecHH()`.

The IMAT horizontal correlation table is stored in **hfecQQ**. This table has dimensions **nveclev** vertical levels by **nQQtab** values of $1-\tau$. The tabulated values of $1-\tau$ are divided into two ranges of uniformly spaced values:

- A fine-scale range of **nQQtb1** values with spacing 1/**qxQtb1** and $0 \leq 1-\tau <$ **QQbeg2**.
- A coarse-scale range of **nQQtb2** values with spacing 1/**qxQtb1** and **QQbeg2** $\leq 1-\tau <$ 1 − **Qcoslim**.

The variables **qxQtb1** and **qxQtb2** are the inverses of the coarse and fine scale $1-\tau$ increments **QQinc1** and **QQinc2**, respectively.

- The forecast water vapor mixing ratio vertical error correlation function tables and related data are contained in common blocks from **vfecQQ.h**:

```
common/vfecQQ0/ vfecQQ(MXveclev,MXveclev)
```

The vertical correlation IMAT table is stored in **vfecQQ**.

The IMAT tables are calculated using the State I function parameter tables and vertical correlation coefficient tables initialized by **initRSRC()**. The calculation of the IMATs **hfecQQ** and **vfecQQ** is implemented in the routine **set_fecQQ()** (Figure 36).

The State I data for the water vapor mixing ratio forecast error correlations **set_fecQQ()** by **USE** of the modules **hfecQ_tbl** (horizontal) and **vfecQ_tbl** (vertical).

The routine **set_fecQQ()** performs the following steps:

1. The values of **QQbeg2**, **Qcoslim**, **qxQtb1**, and **qxQtb2** are calculated using the **REAL** variables **QQmx1**, **QQmx2**, and **dlim**. The steps in this calculation are:
   (a) Calculation of **QQmx1** and an initial value of **QQmx2**:

```
QQmx2=1.-cos(6030./rade)
QQmx1=.036*QQmx2
```

   where **rade** is the Earth's radius, and enters **set_fecQQ()** through **USE** of the module **config**.
   (b) Determination of the value of **dlim**. The maximum value of the correlation cutoff distance value for each class on each vertical level for which function parameters are defined resides in **pars_hfecQ**. This set of elements is scanned to determine their maximum value, which is assigned to the variable **dlim**:

```
dlim=0.
do lv=1,nlev_hfecQ
  if(pars_hfecQ(1,lv).gt.dlim) dlim=pars_hfecQ(1,lv)
end do
```

   (c) The value of **dlim** is used to calculate **QQmx2**:

```
QQmx2=1.-cos(dlim/rade)
QQmx2=max(3.*QQmx1,QQmx2)
```

   (d) Calculation of the $1-\tau$ increments **QQinc1** and **QQinc2**:

```
QQinc1=QQmx1/nQQtb1
QQinc2=(QQmx2-QQmx1)/nQQtb2
```

   (e) The tabulated $1-\tau$ values are calculated by a call to **intp_ctaus()**:

```
call intp_ctaus(nQQtb1,QQinc1,nQQtb2,QQinc2,nQQtab,ctaus)
```

The tabulated values of $1-\tau$ are returned in `ctaus(1:nQQtb1+nQQtb2)`.

(f) Determination of the variables `QQbeg2` and `Qcoslim` from the array `ctaus`:

```
QQbeg2=ctaus(nHHtb1+1)
Qcoslim=1.-ctaus(nHHtab)
```

(g) Calculation of the values of `qxQtb1` and `qxQtb2`:

```
qxQtb1=1./QQinc1
qxQtb2=1./QQinc2
```

2. Once the $1-\tau$ ranges and increments have been determined, The horizontal IMAT interpolation is performed by a call to `intp_hCor()`:

```
call intp_hCor(name_hfecQ,                          &
               type_hfecQ,nlev_hfecQ,plev_hfecQ, &
               MXpar_hc,  npar_hfecQ,pars_hfecQ, &
               nveclev,pveclev, nQQtab, ctaus,   &
               0,MXveclev,hfecQQ,hfecQQ,hfecQQ    ) .
```

The argument `hfecQQ` is repeated three times in this call. This is to retain a common interface to `intp_hCor()`. The second and third appearances of `hfecQQ` are normally used to calculate the first and second derivatives of the IMAT with respect to $1-\tau$. Since these derivatives are not needed in the calculation of C^h (as indicated by the `0` in the argument list to `intp_hCor()`), only the horizontal correlation function table is returned in `hfecQQ(1:nQQtab,1:nveclev)`.

3. The table `hfecQQ` is normalized for use in non-separable correlation functions by division by two:

```
hfecQQ(1:nveclev,1:nQQtab) = .5 * hfecQQ(1:nveclev,1:nQQtab)
```

4. The vertical correlation table is calculated by a call to `intp_vCor()`:

```
call intp_vCor('vfecQQ',lvmax_vc,nlev_vfecQ,plev_vfecQ,   &
               corr_vfecQ,MXveclev,nveclev,pveclev,vfecQQ  )
```

The coarse-grained vertical correlation coefficient table is contained in `corr_vfecQ`, which is defined for the `nlev_vfecQ` pressure levels stored in `plev_vfecQ`. The vertical correlation coefficient IMAT is `vfecQQ`, which is defined for the IMAT `nveclev` pressure levels stored in `pveclev`.

7.4 Observation Error Univariate Correlation Tables — `set_oecHH()`

The elements of the dense matrices C_u^o and C_c^o are obtained by the grid evaluation of the observation error correlation functions. The matrix C_u^o requires only a set of vertical correlation coefficients, while C_c^o requires vertical correlation coefficients and values of observation error horizontal correlation functions. These quantities are all stored in State II IMAT tables.

The State II data for C_u^o and C_c^o are:

- The observation error horizontal correlation function data structures defined in the include file `hoecHH.h`:

```
common /hoecHH0/ hoecHH(MXveclev,nHHotab,MX_hoecH)
common /hoecHH1/ 0coslim(MX_hoecH),00beg2(MX_hoecH)
common /hoecHH2/ qx0tb1(MX_hoecH),qx0tb2(MX_hoecH)
```

  - The horizontal correlation IMAT table is **hoecHH**, which is organized by pressure level, polarity index τ, and function class. The variables **0coslim**, **00beg2**, **qx0tb1**, and **qx0tb2** are discussed later in this section.

- The vertical correlation function IMAT table is **voecHH**, which is organized by vertical level and table class. It is defined in the include file **voecHH.h**:

```
common/voecHH0/ voecHH(MXveclev,MXveclev,MX_voecH)
```

The observation error correlation IMATs are calculated by the routine **set_oecHH()** (Figure 36). The State I data for the observation error correlations enter **set_oecHH()** via **use** of the modules **hoecH_tbl** (horizontal) and **voecH_tbl** (vertical). The IMAT vertical levels are input through inclusion of the include file **levtabl.h**.

The univariate observation error horizontal correlations are functions of the variable $1-\tau$. Let **i** $= 1, \ldots,$ **n_hoecH** be the index of a given observation error horizontal correlation IMAT. There are two ranges of tabulated values of $1-\tau$ in each observation error horizontal correlation IMAT:

- A fine-scale range of **n00tb1(i)** values with spacing $1/$**qx0tb1(i)**, corresponding to $0 \leq 1-\tau <$ **00beg2(i)**.
- A coarse-scale range of **nHHtb2(i)** values with spacing $1/$**qx0tb2(i)**, corresponding to **00beg2(i)** $\leq 1-\tau < 1-$ **0coslim(i)**.

The routine **set_oecHH()** performs the following steps:

1. The calculation of the **n_hoecH** observation error horizontal correlation function tables is implemented in a loop over the index **i = 1,n_hoecH**. Each value of **i** corresponds to a function class. For each value of **i**, the following steps are taken:
   (a) The values of **00beg2(i)**, **0coslim(i)**, **qx0tb1(i)**, and **qx0tb2(i)** are calculated using the **REAL** variables **HHmx1**, **HHmx2**, **00inc1**, **00inc2**, **corlen**, and **dlim**. The steps in this calculation are:
      i. Calculation of the value of **corlen**:
```
corlen=0.
do lv=1,nlev_hoecH(i)
  corlen=corlen+pars_hoecH(2,lv,i)
end do
corlen=corlen/nlev_hoecH(i)
```
      ii. Calculation of **00inc1**:
```
HHomx1=1.-cos(2.*corlen/rade)
00inc1=HHomx1/n00tb1
```
      iii. Calculation of **00inc2**:
```
dlim=0.
do lv=1,nlev_hoecH(i)
  if(pars_hoecH(1,lv,i).gt.dlim) dlim=pars_hoecH(1,lv,i)
end do
HHomx2=1.-cos(dlim/rade)
if(HHomx2.le.HHomx1) HHomx2=1.-cos(8.*corlen/rade)
00inc2=(HHomx2-HHomx1)/n00tb2
```

iv. Calculation of tabulated $1-\tau$ values for this observation error class. This is accomplished by a call to `ctaus()`:

```
call intp_ctaus(nOOtb1,OOinc1,nOOtb2,OOinc2,nHHotab,ctaus)
```

The spacings for the fine and coarse $1-\tau$ ranges are returned in `OOinc1` and `OOinc2`, respectively.

v. Calculation of `qxOtb1(i)` and `qxOtb2(i)` from `OOinc1` and `OOinc2`:

```
qxOtb1(i)=1./OOinc1
qxOtb2(i)=1./OOinc2
```

(b) The observation error horizontal correlation IMAT table for error class `i` is calculated for IMAT level and tabulated $1-\tau$ values using a call to `intp_hCor()`:

```
call intp_hCor(name_hoecH(i),                                    &
               type_hoecH(i),nlev_hoecH(i),plev_hoecH(1,i),      &
               MXpar_hc,     npar_hoecH(i),pars_hoecH(1,1,i),&
               nveclev,pveclev, nHHotab, ctaus,                  &
               0,MXveclev,hoecHH(1,1,i),hoecHH(1,1,i),           &
               hoecHH(1,1,i)                                     ) .
```

The argument `hoecHH` is repeated three times in this call. This is done to retain a common interface to `intp_hCor()`. All of the observation error correlation functions are univariate, and there is no need to calculate the derivatives with respect to $1-\tau$ that were necessary for the forecast error correlation functions. The '0' in the agrument list to `intp_hCor()` (preceding `MXveclev`) indicates that derivatives are not calculated.

2. The observation error vertical correlation coefficient IMAT structure `voecHH` is composed of `n_voecH` tables, which are calculated by `n_voecH` calls to `intp_vCor()`, one for each observation error class:

```
do i=1,n_voecH
   call intp_vCor(name_voecH(i),                              &
                  lvmax_vc,nlev_voecH(i),plev_voecH(1,i), &
                  corr_voecH(1,1,i),                          &
                  MXveclev,nveclev,pveclev, voecHH(1,1,i)  )
end do
```

The coarse-grained observation error vertical correlation coefficient tables reside in the array `corr_voecHH`, and for each class the pressure levels are defined by `plev_voech(1:nlev_voecH(i),i)`. These values are interpolated to tables whose dimensions are `nveclev` by `nveclev` entries, corresponding to pressure levels contained in the array `pveclev`. The final product is the IMAT structure `voecHH`.

7.5 Calculation of IMAT Tables for α_{um}, α_{ul}, α_{vm}, and α_{vl} — `imat_alpha()`

The geostrophic balance parameters α_{um}, α_{ul}, α_{vm}, and α_{vl} that appear in Γ^h are stored in the IMAT listed in Table 16. These IMAT tables have dimensions the `nveclat` latitudes by `nveclev` pressure levels.

The calculation of these IMAT tables is implemented in the subroutine `imat_alpha()` (Figure 37). The IMAT tables enter `imat_alpha()` — and are shared as needed by the PSAS — by `USE` of the module `FEalpha_imat`. The table latitudes enter via `USE` of the module `rlat_imat`.

The PSAS supports a number of models for α_{um}, α_{ul}, α_{vm}, and α_{vl}. The model used is determined by the `CHARACTER` tag `FEalpha_type`, which was read from the resource file during the resource initialization by the subroutine `tabl_FEalpha()`. The routine `imat_alpha()`

Table 16: Objects from the module `FEalpha_imat`

IMAT Structure	Quantity
`Aum_imat`	α_{um}
`Avm_imat`	α_{vm}
`Aul_imat`	α_{ul}
`Avl_imat`	α_{vl}

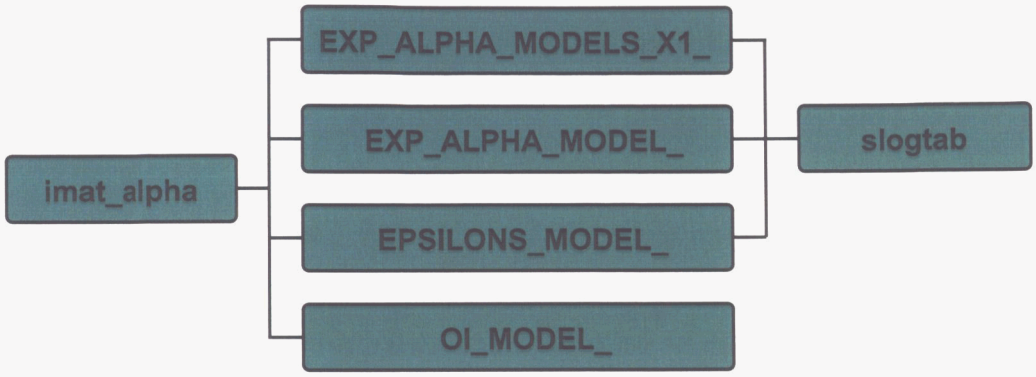


Figure 37: Calling tree for `imat_alpha()`.

comprises a `select case(FEalpha_type)` block, with each case resulting in a call to an internal procedure that calculates the α IMATs:

- `case('PSAS:X1')`: A call is issued to the internal routine `EXP_ALPHAS_model_x1()`
- `case('EXP-ALPHAS','PSAS:XO')`: A call is issued to the internal routine `EXP_ALPHAS_model_`
- `case('EPSILONS')`: A call is issued to the internal routine `EPSILONS_model_`
- `case('GEOS/DAS-OI')`: A call is issued to the internal routine `OI_model_`

The models implemented in the routines `EXP_ALPHAS_model_x1()`, `EXP_ALPHAS_model_`, and `EPSILONS_model_` have level-dependent parameters. Calculation of the IMATs using these models involves log-linear vertical interpolation.

7.6 IMAT Tables for σ^{ψ} and σ^{χ} — `imat_sigW()`

The calculation of Σ^{ψ}, Σ^{χ} and their transposes is facilitated by IMAT tables of σ^{ψ} and σ^{χ} (Table 17). The calculation of the IMATs for σ^{ψ} and σ^{χ} is implemented in the subroutine `imat_sigW()` (Figure 38).

The IMAT tables enter `imat_sigW()` — and are shared where needed by the PSAS — by `USE` of the module `FEsigW_imat`. The table latitudes enter by `USE` of the module `rlat_imat`. The IMAT levels are accessed by inclusion of the header file `levtabl.h`. Choice of the type of model used to calculate σ^{ψ} and σ^{χ} and tabulated values of the σ^{ψ} and σ^{χ} enter via the module `FEsigW_tabl`.

Table 17: IMATs for σ^{ψ} and σ^{χ}

IMAT Structure	Quantity
FEsigS_imat	σ^{ψ}
FEsigV_imat	σ^{χ}

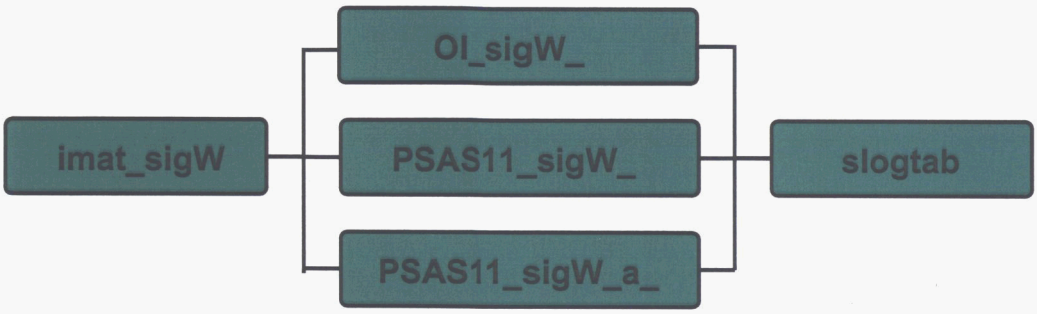


Figure 38: Calling tree for `imat_sigW()`.

The choice of algorithm used to calculate σ^{ψ} and σ^{χ} is controlled by a `select case` block over the `CHARACTER` tag variable `FEsigW_type`, whose value was read from the resource file during the initialization routine `tabl_FEsigW()`. The choices are:

- `case ('PSAS1.1A')`: The quantities σ^{ψ} and σ^{χ} are only level dependent. The calculation is carried out by the internal routine `PSAS11_sigW_a()`. Log-linear vertical interpolation is used to calculate σ^{ψ} and σ^{χ} on IMAT levels.
- `case (' ','PSAS1.1')`: The quantities σ^{ψ} and σ^{χ} are only level dependent. The calculation is carried out by the internal routine `PSAS11_sigW_()`. Log-linear vertical interpolation is used to calculate σ^{ψ} and σ^{χ} on IMAT levels.
- `case ('GEOS/DAS-OI')`: Optimal Interpolation option. The calculation is carried out by the internal routine `OI_sigW_()`. Log-linear vertical interpolation is used to calculate σ^{ψ} and σ^{χ} on IMAT levels.
- `case ('NULL')`: Both σ^{ψ} and σ^{χ} are zero:

```
FEsigS_imat(1:nveclev,1:nveclat)=0.
FEsigV_imat(1:nveclev,1:nveclat)=0.
```

7.7 Forecast Error Standard Deviations

7.7.1 Input of Gridded Forecast Error Standard Deviation Data — `getsigF()`

The PSAS calculates forecast error standard deviations using the sea-level pressure error standard deviation $\sigma^{p_{sl}}$, the upper-air geopotential height error standard deviation σ^{h}, and the upper-air water vapor mixing ratio standard deviation σ^{q}. The PSAS reads the σ^{h} and σ^{q} fields from a GrADS binary file, and calculates $\sigma^{p_{sl}}$ from σ^{h} using the subroutine `getsigF()` (Figure 39):

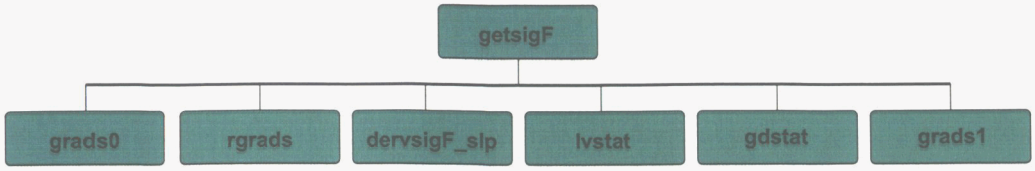


Figure 39: Calling tree for `getsigF()`.

```
subroutine getsigF(iaidim,jaidim,nlevai,plevai, &
                   slp_sigF,HH_sigF,qq_sigF      )
```

The arguments to `getsigF()` are summarized in Table 18.

The routine `getsigF()` performs the following steps to determine $\sigma^{p_{sl}}$, σ^h, and σ^q:

1. The name of the GrADS control file is determined from the PSAS resource file. A Fortran I/O device is assigned to the control file.

2. The contents of the control file are read using a call to subroutine `grads0()`:

   ```
   call grads0(ier,tabln,isigFdim,jsigFdim,ksigFdim,lsigFdim)
   ```

   The input name of the control file is the `CHARACTER` variable `tabln`. This call returns the attributes of the GrADS data file: the number of latitude bands `isigFdim`, the number of longitudes `jsigFdim`, the number of vertical levels `ksigFdim`, and the number of variables `lsigFdim`. The dimensions `isigFdim`, `jsigFdim`, and `ksigFdim` are checked against the analysis grid dimensions `iaidim`, `jaidim`, and `nlevai` respectively, for compatibility (`getsigF()` exits if differences are found in the two sets of dimensions). The `INTEGER` variable `ier` is an error flag, and is zero if no error occurred.

3. Individual 2-D horizontal slices of the fields σ^h and σ^q are read using calls to the routine `rgrads()`.

4. The geopotential height error standard deviation σ^h field corresponding to sea level is read into the array `slp_sigF` via a call to `rgrads()`. The sea-level pressure error standard deviation field $\sigma^{p_{sl}}$ is calculated by a call to `dervsigF_slp()`:

   ```
   call dervsigF_slp(iaidim,jaidim,slp_sigF,slp_sigF)
   ```

   The `INTEGER` arguments `iaidim` and `jaidim` are the horizontal dimensions of the analysis grid. The variable `slp_sigF` appears twice in this call; the first argument is the `INPUT` σ^h values, the second the `OUTPUT` $\sigma^{p_{sl}}$ values.

5. Diagnostic output of the fields' statistics are calculated and written to `stdout` using calls to the routines `lvstat()` and `gdstat()`.

6. The input files are closed by a call to `GRADS1()`.

The gridded forecast error standard deviation fields $\sigma^{p_{sl}}$ (`slp_sigF`), σ^h (`HH_sigF`), and σ^q (`qq_sigF`) are returned to the calling routine through the interface.

Table 18: Data passed into `getsigF()` via its interface.

Variable	Type	Intent	Description
`iaidim`	`INTEGER`	`IN`	Number of grid latitudes
`jaidim`	`INTEGER`	`IN`	Number of grid longitudes
`nlevai`	`REAL(nobslat)`	`IN`	Number of grid vertical levels
`plevai`	`INTEGER`	`IN`	Pressure levels
`slp_sigF`	`REAL(iaidim,jaidim,nlevai)`	`OUT`	Forecast error $\sigma^{p_{sl}}$
`HH_sigF`	`REAL(iaidim,jaidim,nlevai)`	`OUT`	Forecast error σ^{h}
`qq_sigF`	`REAL(iaidim,jaidim,nlevai)`	`OUT`	Forecast error σ^{q}

7.7.2 Gridded Fields of the Elements of Σ^h — `dervsigF_slD()`

The diagonal sea-level windfield elements of Σ^h are the standard deviations of the latitudinal (φ) and longitudinal (λ) gradients of the sea-level pressure errors $\epsilon^{p_{sl}}$. These quantities are currently given by

$$\mathrm{Var}(\partial_m \epsilon^{p_{sl}}) = \mathrm{Var}(\partial_l \epsilon^{p_{sl}}) = \left[\frac{\sigma^{p_{sl}} \sqrt{\rho^{h'}(1)}}{2\Omega \bar{\rho} a} \right]^2 , \tag{11}$$

where $\bar{\rho}$ is mean sea-level air density, Ω is the Earth's rotational angular frequency, a is the Earth's radius, and $\mathrm{Var}(\cdot)$ is the variance. This calculation is implemented in subroutine `dervsigF_slD()` (Figure 40), whose arguments are summarized in Table 19:

```
subroutine dervsigF_slD(im,jnp,sigF_ps,sigF_pm,sigF_pl)
```

The physical constants $\bar{\rho}$, Ω, and a are stored in the variables `rhobar`, `omega`, and `r_earth`. The aforementioned physical constants enter `dervsigF_slD()` through `USE` of the module `const`. The factor $\sqrt{\rho^{h'}(1)}$ is the inverse of the quantity stored in the IMAT `norm_DD`, which enters `dervsigF_slD()` through the include file `vfecHH.h`. The IMAT table pressure levels `pveclev(1:nveclev)` are input through `levtabl.h`.

The calculation entails the vertical interpolation of IMAT values from `norm_DD` to the sea-level value pressure value, and use of this interpolated value of $\sqrt{\rho^{h'}(1)}$ to calculate the diagonal elements, which are returned in the arrays `sigF_pm(1:im,1:jnp)` and `sigF_pl(1:im,1:jnp)`.

The procedure for calculating `sigF_pm` and `sigF_pl` is outlined below:

1. Index the sea-level pressure `pres4slp` against the IMAT pressure levels by calling `slogtab()`:

   ```
   call slogtab(.not.nearest, nveclev,pveclev, 1, pres4slp, lv,wt)
   ```

   The IMAT level index is returned in `lv`, the log-linear interpolation weight in `wt`.

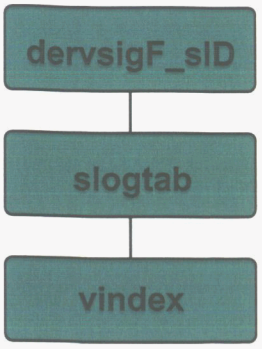


Figure 40: Calling tree for `dervsigF_slD()`.

Table 19: Data passed into `dervsigF_slD()` via its interface.

Variable	Type	Intent	Description
`im`	`INTEGER`	`IN`	Number of grid latitudes
`jnp`	`INTEGER`	`IN`	Number of grid longitudes
`sigF_ps`	`REAL(im,jaidim)`	`OUT`	Forecast error $\sigma^{p_{sl}}$
`sigF_pm`	`REAL(im,jnp)`	`OUT`	$\sqrt{\mathrm{Var}\left(\partial_m \epsilon^{p_{sl}}\right)}$
`sigF_pl`	`REAL(im,jnp)`	`OUT`	$\sqrt{\mathrm{Var}\left(\partial_l \epsilon^{p_{sl}}\right)}$

2. The normalization factor `norm` is calculated from the IMAT `norm_DD` by log-linear interpolation:

```
norm = norm_DD(lv,kmat_HGHT) + wt * (norm_DD(lv+1,kmat_HGHT) &
                                 -  norm_DD(lv,kmat_HGHT)
```

   The `INTEGER` index `kmat_HGHT` indicates the h-portion (mass-coupled height/winds) of the IMAT `norm_DD`. This index enters `dervsigF_slD()` through `USE` of the module `config`.

3. Calculation of `sigF_pm` and `sigF_pl`:

```
sigF_pm(:,:) = sigF_ps(:,:) * scalar / norm
sigF_pl(:,:) = sigF_pm(:,:)
```

   The `REAL` parameter `scalar` is defined from input constants $\bar{\rho}$, Ω, and a:

```
scalar = 100. / (rhobar * 2. * OMEGA * R_EARTH)
```

   The factor of 100 is a pressure units conversion from hPa to Pa.

7.7.3 Gridded Fields of the Elements of Σ^h — `dervsigF_upD()`

The diagonal upper-air windfield elements of Σ^h are the standard deviations of φ and λ gradients of the upper-air geopotential height forecast errors ϵ^h. These quantities are currently

$$\mathrm{Var}(\partial_m \epsilon^h) = \mathrm{Var}(\partial_l \epsilon^h) = \left[\frac{g\sigma^h \sqrt{\rho^{h\prime}(1)}}{2\Omega a} \right]^2 , \tag{12}$$

where g is the acceleration due to gravity. This calculation is implemented in subroutine `dervsigF_upD()` (Figure 41):

```
subroutine dervsigF_upD(im,jnp,mlev,pres_lev, &
                        sigF_H,sigF_Hm,sigF_Hl )
```

The arguments to `dervsigF_upD()` are summarized in Table 20.

The physical constant g is stored in the variable `g_earth`, and enters `dervsigF_upD()` through `USE` of the module `const`. The normalization factor $\sqrt{\rho^{h\prime}(1)}$ is the inverse of the quantity stored in the IMAT `norm_DD`. The include file `vfecHH.h` contains `norm_DD`. The IMAT table pressure levels are stored in the array `pveclev(1:nveclev)`, which is defined in `levtabl.h`.

The calculation entails the vertical interpolation of IMAT values from `norm_DD` to the analysis grid pressure values stored in `pres_lev(1:mlev)`, and use of these interpolated values of $\sqrt{\rho^{h\prime}(1)}$ to calculate the diagonal elements, which are returned in the arrays `sigF_Hm(1:im,1:jnp,1:mlev)` and `sigF_Hl(1:im,1:jnp,1:mlev)`.

This is accomplished through the following steps:

1. Allocation of temporary work arrays `ilev(mlev)` and `wlev(mlev)`, corresponding to IMAT pressure level indices and log-linear interpolation weights, respectively.

2. Index the analysis levels against the IMAT pressure levels by a call to `slogtab()`:

   ```
   call slogtab(.not.nearest,nveclev,pveclev,mlev,pres_lev,ilev,wlev)
   ```

   The input IMAT levels `pveclev(1:nveclev)` enter `dervsigF_upD()` via the module `rlat_imat`. The IMAT level indices are returned in `ilev(1:mlev)`, the log-linear interpolation weights in `wlev(1:mlev)`.

3. Interpolation of IMAT values to the analysis grid levels, and calculation of `sigF_Hm` and `sigF_Hl`:

   ```
   do k=1,mlev
      lv=ilev(k)
      wt=wlev(k)
      norm = norm_DD(lv,kmat_HGHT)
      norm = norm + wt *(norm_DD(lv+1,kmat_HGHT) - norm)
      sigF_Hm(:,:,k) = sigF_H(:,:,k) * scalar / norm
      sigF_Hl(:,:,k) = sigF_Hm(:,:,k)
   end do
   ```

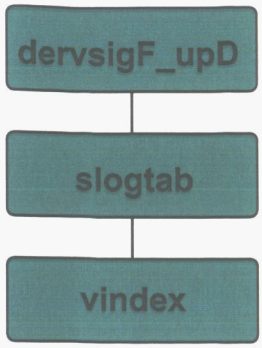


Figure 41: Calling tree for `dervsigF_upD()`.

Table 20: Data passed into `dervsigF_upD()` via its interface.

Variable	Type	Intent	Description
`im`	`INTEGER`	`IN`	Number of grid latitudes
`jnp`	`INTEGER`	`IN`	Number of grid longitudes
`mlev`	`INTEGER`	`IN`	Number of grid levels
`pres_lev`	`REAL(mlev)`	`IN`	Number of grid levels
`sigF_H`	`REAL(im,jnp,mlev)`	`OUT`	Forecast error σ^h
`sigF_Hm`	`REAL(im,jnp,mlev)`	`OUT`	$\sqrt{\mathrm{Var}\left(\partial_m \epsilon^h\right)}$
`sigF_Hl`	`REAL(im,jnp,mlev)`	`OUT`	$\sqrt{\mathrm{Var}\left(\partial_l \epsilon^h\right)}$

The `INTEGER` index `kmat_HGHT` indicates the h-portion (mass-coupled height/winds) of the IMAT `norm_DD`. This index enters `dervsigF_upD()` through `USE` of the module `config`. The quantity `scalar` is a REAL parameter equal to $g/2\Omega a$:

```
scalar = G_EARTH/(2.*OMEGA*R_EARTH)
```

4. Deallocation of the temporary arrays `ilev` and `wlev`.

The results are returned in the arrays `sigF_Hm(1:im,1:jnp,1:mlev)` and `sigF_Hl(1:im,1:jnp,1:mlev)`.

7.7.4 Interpolation of Elements of Σ^h — `intp_sigF()`

The elements of Σ^h corresponding to forecast geopotential height, sea-level pressure, and upper-air water-vapor mixing ratio — σ^h, $\sigma^{p_{sl}}$, and σ^q were read and calculated through a call to `getsigF()`. Gridded fields of the elements of Σ^h corresponding to the wind components were calculated by calling `dervsigF_slD()` and `dervsigF_upD()`.

The solution of the Innovation Equation (1) requires the elements of Σ^h be evaluated at observation points. The calculation of the analysis increments (AI), Eqn. (2), requires

values of the elements of Σ^h at both observation and QEA grid points. The calculation of the elements of Σ^h at the observation and QEA grid locations is accomplished via interpolation, implemented in subroutine **intp_sigF()** (Figure 42):

```
subroutine intp_sigF( im, jnp, mlev, pres_lev,           &
                      sigF_ps, sigF_pm, sigF_pl,         &
                      sigF_H,  sigF_Hm, sigF_Hl, sigF_q, &
                      n_kr, kr_loc, kr_len,              &
                      n_kt, kt_loc, kt_len,              &
                      n_x, rlat,rlon,rlev, sigF           )
```

The arguments to **intp_sigF()** are summarized in Table 21.

The routine **intp_sigF()** performs interpolation of analysis grid data to locations defined by the attribute vectors **rlat**, **rlon**, and **rlev**. Two types of interpolation are perfomed:

- Horizontal interpolation of sea-level pressure (**case(ktps)** below) and winds (**case(ktpm,ktHp)** below) to sea-level locations listed in **rlat**, **rlon**, and **rlev**. Bilinear interpolation is used.
- Three-dimensional interpolation of upper-air geopotential heights (**case(ktHH)** below), upper-air winds (**case(ktHm,ktHl)** below), and upper-air water-vapor mixing ratio (**case(ktqq)** below) to locations listed in the arrays **rlat**, **rlon**, and **rlev**. The interpolation is performed using horizontal bilinear interpolation, followed by vertical log-linear interpolation.

The input analysis grid data are stored in the arrays **sigF_ps**, **sigF_pl**, **sigF_pm**, **sigF_H**, **sigF_Hl**, **sigF_Hm**, and **sigF_q**. The **n_x** output forecast error standard deviations are returned in the array **sigF**.

The interpolation proceeds through the following steps:

1. The initialization steps listed below:
   (a) Allocation of temporary work arrays: vertical level index for the output grid **ilev(mlev,n_x)** and log-linear interpolation weights **wlev(mlev,n_x)**; nearest grid latitude index **ilat(n_x)**, and linear interpolation weight **wlat(n_x)**; nearest grid longitude index **ilon(n_x)** and linear interpolation weight **wlon(n_x)**.
   (b) Computation of the nearest grid longitude index **ilon** and grid latitude index **ilat** for each output grid location:

```
do i=1,n_x
   ilon(i) = int((rlon(i)+180.)/dlon)
   ilat(i) = max(1,min(int((rlat(i)+90.)/dlat)+1,jnp-1))
end do
```

   The output grid location referenced by index **i** has longitude λ between **ilon(i) * dlon -180.** and **(ilon(i) + 1) * dlon -180.**, and latitude φ between **ilat(i) * dlat - 90.** and **(ilat(i) + 1) * dlat - 90.**
   (c) Computation of the bilinear horizontal interpolation weights for each output grid location:

```
do i=1,n_x              ! with linear weights
  wlon(i)=(rlon(i)+180.)/dlon -  ilon(i)
  wlat(i)=(rlat(i)+ 90.)/dlat - (ilat(i)-1)
end do
```

(d) Index the output grid pressure level values `rlev(1:n_x)` against the grid pressure levels `pres_lev(1:mlev)` by calling `slogtab()`:

```
call slogtab(.not.nearest,mlev,pres_lev,n_x,rlev,ilev,wlev)
```

The index corresponding to the higher pressure level is returned in `ilev(1:mlev,1:n_x)`. The log-linear interpolation weights in are returned in `wlev(1:mlev,1:n_x)`.

2. The interpolation is carried out as a loop over the `kr/kt` segments in the output grid vector. The interpolation steps are:

(a) For each `kr/kt` segment, the beginning index `lc`, ending index `le`, and length `ln` are determined:

```
lc=kr_loc(kr)+kt_loc(kt,kr)
ln=kt_len(kt,kr)
le=lc+ln-1
```

(b) If `ln` $\neq 0$, interpolation must be performed for this `kr/kt` segment. The type of calculation is governed by a `select case(kt)` block. The constant `kt`-values against which each value of `kt` is tested are input from the module `config`:

- `case(ktps)`: This corresponds to sea-level pressure values. The interpolation in this case is bilinear:

```
do i=lc,le
   sigF(i)=bilinear_(sigF_ps,ilon(i),wlon(i),ilat(i),wlat(i))
end do
```

- `case(ktpm,ktpl)`: This corresponds to the sea-level windfield components. Again, this is bilinear interpolation:

```
do i=lc,le
   sigF(i)=bilinear_(sigF_pm,ilon(i),wlon(i),ilat(i),wlat(i))
end do
```

Note that this assumes that `sigF_pl` = `sigF_pm`, which is currently what is produced by `dervsigF_slD()` (see Section 7.7.2).

- `case(ktHH)`: Upper-air geopotential heights. This is horizontal bilinear interpolation followed by vertical log-linear interpolation. The vertical index array `ilev` determines which grid levels lie immediately above and below the output grid point. Bilinear interpolation is used to calculate the values of `sigF` directly above and below the output grid location. The vertical log-linear interpolation is performed by combining these two values using the log-linear interpolation weight value from `wlev`:

```
do i=lc,le
   sigF(i)= bilinear_(sigF_H(1,1,ilev(i)),          &
                      ilon(i),wlon(i),ilat(i),wlat(i) )
   if(ilev(i).lt.mlev) then
      sigm=bilinear_(sigF_H(1,1,ilev(i)+1),         &
                      ilon(i),wlon(i),ilat(i),wlat(i) )

      sigF(i)=sigF(i) + wlev(i)*(sigm-sigF(i))
   endif
end do
```

- `case(ktHm,ktHl)`: The upper-air winds. Both fields are calculated from gridded values of `sigF_Hm`. This is compatible with the current version of `dervsigF_upD()` (see Section 7.7.3).
- `case(ktqq)`: The upper-air water vapor mixing ratio. Segments of `sigF` corresponding to σ^q are calculated using the gridded values of σ^q stored in the array `sigF_q`.

3. Upon completion of the interpolation, the temporary arrays `ilev`, `wlev`, `ilat`, `wlat`, `ilon`, and `wlon` are deallocated.

Table 21: Data passed into `intp_sigF()` via its interface.

Variable	Type	Intent	Description
im	INTEGER	IN	Number of grid latitudes
jnp	INTEGER	IN	Number of grid longitudes
mlev	INTEGER	IN	Number of grid levels
pres_lev	REAL(mlev)	IN	Number of grid levels
sigF_ps	REAL(im,jaidim)	OUT	Forecast error $\sigma^{p_{sl}}$
sigF_pm	REAL(im,jnp)	OUT	$\sqrt{\text{Var}\left(\partial_m \epsilon^{p_{sl}}\right)}$
sigF_pl	REAL(im,jnp)	OUT	$\sqrt{\text{Var}\left(\partial_l \epsilon^{p_{sl}}\right)}$
sigF_H	REAL(im,jnp,mlev)	OUT	Forecast error σ^h
sigF_Hm	REAL(im,jnp,mlev)	OUT	$\sqrt{\text{Var}\left(\partial_m \epsilon^h\right)}$
sigF_Hl	REAL(im,jnp,mlev)	OUT	$\sqrt{\text{Var}\left(\partial_l \epsilon^h\right)}$
sigF_q	REAL(im,jnp,mlev)	OUT	σ^q
n_kr	INTEGER	IN	Number of Regions
kr_loc	INTEGER(n_kr)	IN	Region Starting Indices
kr_len	INTEGER(n_kr)	IN	Region Lengths
n_kt	INTEGER	IN	Number of Data types
kt_loc	INTEGER(n_kt,n_kr)	IN	Starting Indices of `kr`/`kt` segments
kt_len	INTEGER(n_kt,n_kr)	IN	Length of `kr`/`kt` segments
n_x	INTEGER	IN	Number of output grid points
rlat	REAL(n_x)	IN	Output grid latitude φ
rlon	REAL(n_x)	IN	Output grid longitude λ
rlev	REAL(n_x)	IN	Output grid pressure (hPa)
sigF	REAL(n_x)	OUT	Elements of Σ^h at output grid locations

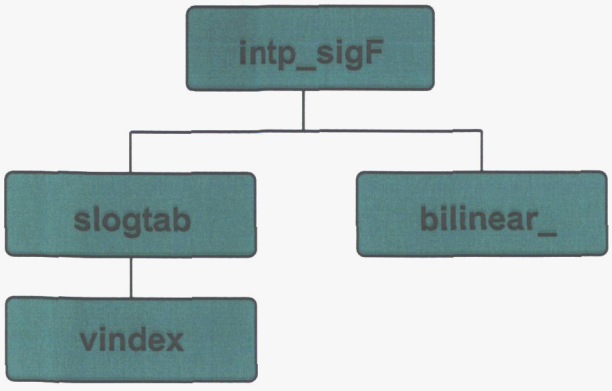


Figure 42: Calling tree for `intp_sigF()`.

8 Stage III: Solution of the Innovation Equation — `solve4x()`

8.1 Control Flow through `solve4x()`

The innovation equation (1) is solved for $\mathbf{x}$ using a nested preconditioned conjugate gradient algorithm described in [da Silva and Guo, 1996]. Figure 44 shows the structure of the matrix $HP^fH^T + R$ for the global solver and its three levels of preconditioning.

The driver routine for the nested preconditioned CG solver is the routine **`solve4x()`**:

```
subroutine solve4x(n_kr,kr_loc,kr_len,kt_len,  &
                   nobs,kxX,rlatX,rlonX,rlevX, &
                   sigU,sigO,sigF,             &
                   nvecs,ldD,Dels,ldX,Xvec      )
```

The calling tree for **`solve4x()`** illustrated in Figure 43, and Table 22 is a summary of its arguments.

The conjugate gradient solver and its preconditioners are implemented in the following routines:

- **`conjgr()`**: global conjugate gradient solver
- **`conjgr2()`**: region-diagonal level 2 preconditioner
- **`conjgr1()`**: univariate-diagonal level 1 preconditioner, and its profile-diagonal level 0 preconditioner.

Much of the workspace used in **`solve4x()`** is dynamically allocated and includes:

- **`kt_loc(ktmax,n_kr)`**: Indices of starting points in attribute vectors of **`kr/kt`** segments.
- **`qr_x(nobs)`**, **`qr_y(nobs)`**, and **`qr_z(nobs)`**: Cartesian components of the radial unit vector $\hat{\mathbf{e}}_r$.
- **`ql_x(nobs)`** and **`ql_y(nobs)`**: Cartesian components of the longitudinal unit vector $\hat{\mathbf{e}}_l$.
- **`qm_x(nobs)`**, **`qm_y(nobs)`**, and **`qm_z(nobs)`**: Cartesian components of the meridional unit vector $\hat{\mathbf{e}}_m$.
- **`ktab(nobs)`** and **`wtab(nobs)`**: **`INTEGER`** vertical level index for imat look-up tables and **`REAL`** log-linear interpolation weights, respectively.
- **`jtab(nobs)`** and **`vtab(nobs)`**: **`INTEGER`** polarity index τ_{ij} index for imat look-up tables and **`REAL`** linear interpolation weights, respectively.

There are some preparatory steps preceding the call to **`conjgr()`**:

- Generation of the **`kr/kt`** segment index table **`kt_loc(1:ktmax,1:n_kr)`**. This is accomplished using the input array of **`kr/kt`** segment lengths **`kt_len(1:ktmax,1:n_kr)`**.

- Calculation of the Cartesian components of the unit vectors $(\hat{\mathbf{e}}_r, \hat{\mathbf{e}}_l, \hat{\mathbf{e}}_m)$ through a call to `ll2qvec()`:

```
call ll2qvec(nobs,rlatX,rlonX,qr_x,qr_y,qr_z, &
             qm_x,qm_y,qm_z,ql_x,ql_y          )
```

- Assign the sounding indices `ks` to the innovations by calling the routine `setpix()`:

```
call setpix(nobs,kxX,rlatX,rlonX,ksX)
```

  The values of `ks` are returned in the array `ksX(1:nobs)`.

- Determine the vertical level IMAT indices `ktab(1:nobs)` and log-linear interpolation weights `wtab(1:nobs)` through a call to the routine `slogtab()`:

```
call slogtab(roundoff,nveclev,pveclev,nobs,rlevX,ktab,wtab)
```

- Determine the polarity index τ_{ij} level IMAT indices `jtab(1:nobs)` and linear interpolation weights `vtab(1:nobs)` through a call to the routine `slintab()`:

```
call slintab(roundoff,nveclat,veclats,nobs,rlatX,jtab,vtab)
```

The conjugate gradient solution `x` (`Xvec(1:nobs,1:nvecs)`) is determined by calling `conjgr()`:

```
call conjgr( cgverb(nbandcg),n_kr,kr_loc,kr_len,ktmax,kt_loc, &
             kt_len,nobs, kxX,ksX,ktab,jtab,sigU,sigO,sigF,   &
             qr_x,qr_y,qr_z,qm_x,qm_y,qm_z,ql_x,ql_y,         &
             nvecs,ldD,Dels,ldX,Xvec,ierr                      )
```

The `INTEGER` scalar variable `ierr` is an error flag. If `ierr` $\neq$ `0`, an `conjgr()` has returned with an error.

8.2 The Conjugate Gradient Solver —`conjgr()`

The conjugate gradient solver algorithm and preconditioning strategy used in the current version of the PSAS is described in [da Silva and Guo, 1996]. The basic solver and preconditioning algorithm is

- Solution of the global problem by the routine `conjgr()`. This problem is preconditioned by a `kr`-diagonal (multivariate) problem. The matrix-vector multiplication $(HP^fH^T + R)\mathbf{x}$ is performed by a call to the routine `op_Mx()`.
- Solution of the `kr`-diagonal (level 2) problem by the routine `conjgr2()`. This problem is preconditioned by a `kr`/`kt`-diagonal (univariate) problem. The matrix-vector multiplication $(HP^fH^T + R)\mathbf{x}$ is performed by a call to the routine `op_Mx()`.
- Solution of the `kr`/`kt`-diagonal (level 1) problem by the routine `conjgr1()`. This system is preconditioned by a `ks`-diagonal problem. The matrix-vector multiplication $(HP^fH^T + R)\mathbf{x}$ is performed by a call to the Basic Linear Algebra Software (BLAS) routine `SSPMV()`. The solution of the `ks`-diagonal (level 0) problem is accomplished by a Cholesky solver, implemented in a BLAS call to the routines `SPPTRF` and `SPPTRS`.

The levels of preconditioning are illustrated in Figure 44. The matrix-vector multiplications required in the solution of the global and level 1 and level 2 preconditioners are implemented in calls to the routine `op_Mx()`, which is described in Section 8.3.

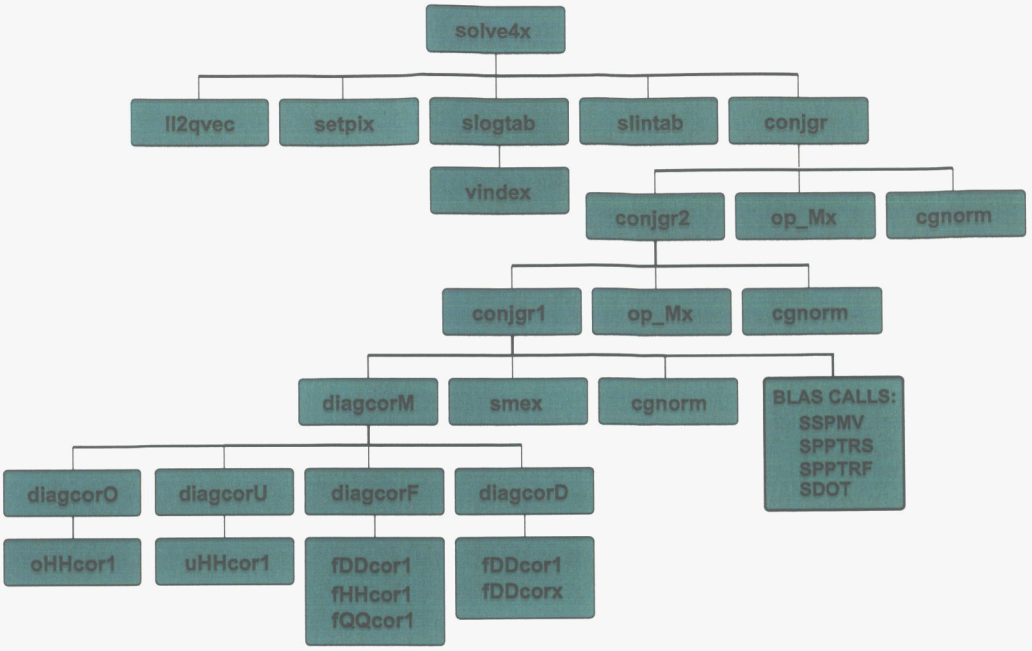


Figure 43: Calling tree for the routine `solve4x()`.

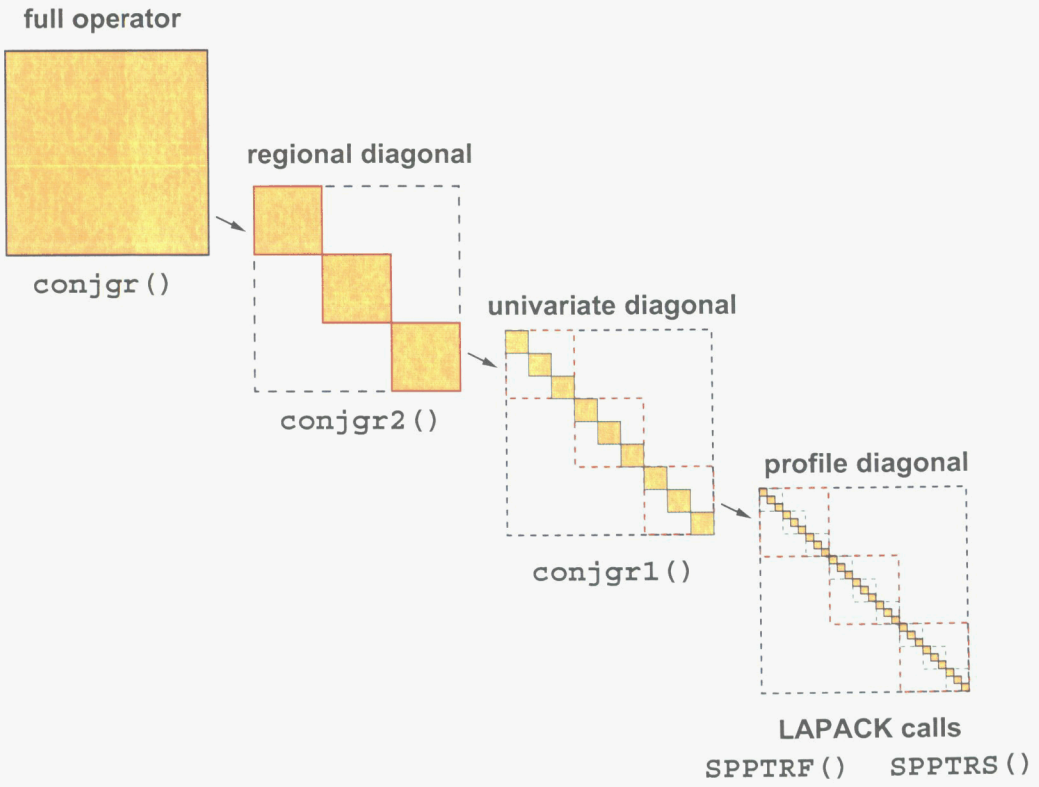


Figure 44: Levels of the nested preconditioner.

Table 22: Summary of arguments to `solve4x()`.

Variable	Type/Dimensions	Intent	Description
n_kr	INTEGER	IN	Number of Regions
kr_loc	INTEGER(n_kr)	IN	Region Starting Indices
kr_len	INTEGER(n_kr)	IN	Region Lengths
kt_len	INTEGER(ktmax,n_kr)	IN	Length of kr/kt Segments
nnobs	INTEGER	IN	Number of Innovations
kxX	INTEGER(nnobs)	IN	Instrument Index kx
rlatX	REAL(nnobs)	IN	Latitude φ
rlonX	REAL(nnobs)	IN	Longitude λ
rlevX	REAL(nnobs)	IN	Pressure (hPa)
sigU	REAL(nnobs)	IN	Obs. Error σ_{ou}
sigO	REAL(nnobs)	IN	Obs. Error σ_{oc}
sigF	REAL(nnobs)	IN	Forecast Error σ_f
nvecs	INTEGER	IN	Number of Vectors
ldD	INTEGER	IN	Innovation Vector Leading dimension
Dels	REAL(ldD,nvecs)	IN	Innovations $\mathbf{w}^o - H\mathbf{w}^f$
ldX	INTEGER	IN	Leading dimension of Xvec ($\mathbf{x}$)
Xvec	REAL(ldX,nvecs)	IN	Intermediate vector ($\mathbf{x}$)

8.3 Software Implementation of $(HP^fH^T + R)\mathbf{x}$ —op_Mx()

The matrix-vector multiplication $(HP^fH^T + R)\mathbf{x}$ for the global solver and preconditioning level 2 is performed by the routine `op_Mx()`. The calling tree for `op_Mx()` is illustrated in Figure 45, and its interface is presented below:

```
subroutine op_Mx(kind_mat,kind_cov,                                  &
                 n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len,             &
                 n_x,  kx,   ks, ktab, jtab, sigU, sigC, sigF,       &
                 qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y,&
                 nvecs,                                              &
                 ldx,x,                                              &
                 ldCx,Cx                                              )
```

A description of the arguments to `op_Mx()` is given in Table 23. The routine `op_Mx()` computes the elements of $HP^fH^T + R$ and calculates the product $(HP^fH^T + R)\mathbf{x}$. The vector $\mathbf{x}$ and the product $(HP^fH^T + R)\mathbf{x}$ are stored in the variables `x` and `Cx`, respectively.

The calculation of `Cx` proceeds by successive application of lower-level operators to intermediate vectors as described in Section 3.3. These intermediate vectors are represented by the dynamically allocated arrays `Tx(1:n_x,1:nvecs)` and `Ty(1:n_x,1:nvecs)`.

The output result array `Cx(1:n_x,1:nvecs)` is initialized with value zero. The factored-operator calculation of $(HP^fH^T + R)\mathbf{x}$ is outlined below.

The Observation Error Covariance Operator. The operator R is calculated using the factored-operator expansion Eqn. (9) by a call to the internal procedure `covOOxpy_()`:

```
call covOOxpy_(ldx, x, ldCx, Cx)
```

The calculation of $R_U\mathbf{x}$ is described below:

1. Calculate $\mathtt{Tx} = \Sigma_u^o\mathbf{x}$ using a call to `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len,n_kt,kt_loc,kt_len, &
             n_x,sigU,nvecs,ldx,x,n_x,Tx             )
```

2. Set $\mathtt{Ty}(1:\mathtt{n_x}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Ty} = C_u^o\mathtt{Tx}$ using a call to `sym_Cxpy()`:

```
call sym_Cxpy(kind_mat,kind_covU, sparse,                    &
              n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len,        &
              n_x,    kx,   ks, ktab,                        &
              qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y, &
              nvecs, n_x, Tx, n_x, Ty, istat                  )
```

3. Set $\mathtt{Tx}(1:\mathtt{n_x}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Tx} = \Sigma_u^o\mathtt{Ty}$ using a call to `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len,n_kt,kt_loc,kt_len, &
             n_x,sigU,nvecs,n_x,Ty,n_x,Tx            )
```

This result is added to the output array `y`:

```
y(1:n_x,1:nvecs) = y(1:n_x,1:nvecs) + Tx(1:n_x,1:nvecs)
```

The calculation and application of R_C follows a similar scheme:

1. Calculate $\mathtt{Tx} = \Sigma_c^o\mathbf{x}$ using a call to `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len,n_kt,kt_loc,kt_len, &
             n_x,sigC,nvecs,ldx,x,n_x,Tx             )
```

2. Set $\mathtt{Ty}(1:\mathtt{n_x}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Ty} = C_c^o\mathtt{Tx}$ using a call to `sym_Cxpy()`:

```
call sym_Cxpy(kind_mat,kind_covC, sparse,                    &
              n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len,        &
              n_x,    kx,   ks, ktab,                        &
              qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y, &
              nvecs, n_x, Tx, n_x, Ty, istat                  )
```

3. Set $\mathtt{Tx}(1:\mathtt{n_x}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Tx} = \Sigma_c^o\mathtt{Ty}$ using a call to `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len,n_kt,kt_loc,kt_len, &
             n_x,sigC,nvecs,n_x,Ty,n_x,Tx            )
```

This result is added to the output array `y`:

```
y(1:n_x,1:nvecs) = y(1:n_x,1:nvecs) + Tx(1:n_x,1:nvecs)
```

The Forecast Error Covariance Operator. The operator HP^fH^T is calculated using the factored-operator expansion Eqn. (7) using the internal procedure `covFFxpy_()`:

```
call covFFxpy_(ldx, x, ldCx, Cx)
```

The product $\Gamma^h\Sigma^h C^h\Sigma^h\Gamma^{hT}\mathbf{x}$ is calculated as follows:

1. Initialize the temporary array `Tx(1:n_x,1:nvecs)` to zero. Calculate `Tx` $= \Gamma^{hT}\mathbf{x}$ by calling the routine `aj_Alf()`:

```
call aj_Alf(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len, &
            n_x,ktab,jtab, nvecs, ldx, x, n_x, Tx    )
```

2. Calculate `Ty` $= \Sigma^h$`Tx` by calling the routine `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len,        &
    n_x,sigF, nvecs, n_x, Tx, n_x, Ty                       )
```

3. Set `Tx`$(1:\mathtt{n_x}, 1:$`nvecs`$) =$ `0`. Calculate `Tx` $= C^h$`Ty` by calling the routine `sym_Cxpy()`:

```
call sym_Cxpy(kind_mat,kind_covF, sparse,           &
    n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len,         &
    n_x,    kx,   ks, ktab,                         &
    qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y, &
    nvecs, n_x, Ty, n_x, Tx, istat                   )
```

4. Calculate `Ty` $= \Sigma^h$`Tx` by calling the routine `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len,        &
    n_x,sigF, nvecs, n_x, Tx, n_x, Ty                       )
```

5. Set the work array `Tx`$(1:\mathtt{n\_x}, 1:$`nvecs`$) =$ `0`. Calculate `Tx` $= \Gamma^h$`Ty` by calling the routine `mv_Alf()`:

```
call mv_Alf(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len,         &
    n_x,ktab,jtab, nvecs, n_x, Ty, n_x, Tx                  )
```

The term $\Gamma^h\Sigma^h C^h\Sigma^h\Gamma^{hT}\mathbf{x}$ is returned `Tx(1:n_x,1:nvecs)`, which is added to the result $\mathbf{y}$ stored in `y(1:n_x,1:nvecs)`:

```
y(1:n_x,1:nvecs) = y(1:n_x,1:nvecs) + Tx(1:n_x,1:nvecs)
```

The mass-decoupled forecast wind error covariance terms are calculated using a similar approach to that outlined for the mass-coupled forecast wind error covariances. These operators act only on the components of the windfields. The lengths of the segments of $\mathbf{x}$ that contain variables that are either upper-air or surface wind components are stored in a dynamically allocated array `kt_lenW(1:n_kt,1:n_kr)`. The forecast error standard deviations σ^ψ and σ^χ are stored in the `sigW(1:n_x)`, which is also an allocatable array. The windfield segment indexing information is set in `kt_lenW` as follows:

```
kt_lenW(:,:) = 0
kt_lenW(ktus,:) = kt_len(ktus,:)
kt_lenW(ktvs,:) = kt_len(ktvs,:)
kt_lenW(ktuu,:) = kt_len(ktuu,:)
kt_lenW(ktvv,:) = kt_len(ktvv,:)
```

The error covariance operator term $\Gamma^{\psi}\Sigma^{\psi}C^{\psi}\Sigma^{\psi}\Gamma^{\psi T}\mathbf{x}$ is calculated for both the upper-air and surface winds as follows:

1. Set $\mathtt{Tx}(1:\mathtt{n\_x}, 1:\mathtt{nvecs}) = 0$.

2. Initialize the wind error streamfunction standard deviation temporary array $\mathtt{sigW}(1:\mathtt{n\_x}) = 0$. Fill in the necessary elements of `sigW` with their respective values of σ^{ψ} obtained from the indirect matrix structure `FEsigS_imat`. This is implemented as a call to the routine `getivec()`.

```
call getivec(MXveclev,MXveclat,FEsigS_imat,          &
             n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
             n_x, ktab, jtab, sigW                    )
```

3. Calculate $\mathtt{Tx} = \Gamma^{\psi T}\mathbf{x}$ by calling the adjoint routine `aj_Bet()`:

```
call aj_Bet(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
            n_x,ktab,jtab, nvecs, ldx, x, n_x, Tx    )
```

4. Calculate $\mathtt{Ty} = \Sigma^{\psi}\mathtt{Tx}$ by calling `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
    n_x,sigW, nvecs, n_x, Tx, n_x, Ty                 )
```

5. Set $\mathtt{Tx}(1:\mathtt{n_x}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Tx} = C^{\psi}\mathtt{Ty}$. This is accomplished by calling the routine `sym_Cxpy()`:

```
call sym_Cxpy(kind_mat,kind_covS, sparse,          &
    n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW,       &
    n_x,    kx,   ks, ktab,                        &
    qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y, &
    nvecs, n_x, Ty, n_x, Tx, istat                 )
```

6. Calculate $\mathtt{Ty} = \Sigma^{\psi}\mathtt{Tx}$ by calling the routine `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
    n_x,sigW, nvecs, n_x, Tx, n_x, Ty                 )
```

7. Set the work array $\mathtt{Tx}(1:\mathtt{n\_x}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Tx} = \Gamma^{\psi}\mathtt{Ty}$ by calling the routine `mv_Bet()`:

```
call mv_Bet(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
    n_x,ktab,jtab, nvecs, n_x, Ty, n_x, Tx           )
```

The term $\Gamma^{\psi}\Sigma^{\psi}C^{\psi}\Sigma^{\psi}\Gamma^{\psi T}\mathbf{x}$ is stored in the array `Tx(1:n_x,1:nvecs)`, which is added to the result $\mathbf{y}$ stored in `y(1:n_x,1:nvecs)`:

```
y(1:n_x,1:nvecs) = y(1:n_x,1:nvecs) + Tx(1:n_x,1:nvecs)
```

The final error covariance operator term $\Gamma^x \Sigma^x C^x \Sigma^x \Gamma^{xT} \mathbf{x}$ is calculated for both the upper-air and surface windfields as follows:

1. Set $\mathtt{Tx}(1:\mathtt{n\_x}, 1:\mathtt{nvecs}) = 0$.
2. Initialize the wind error velocity potential standard deviation temporary array $\mathtt{sigW}(1:\mathtt{n\_x}) = 0$. Fill in the necessary elements of `sigW` with their respective values of σ^x obtained from the indirect matrix structure `FEsigV_imat`. This is implemented as a call to the routine `getivec()`.

```
call getivec(MXveclev,MXveclat,FEsigV_imat,          &
             n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
             n_x, ktab, jtab, sigW                    )
```

3. Calculate $\mathtt{Tx} = \Gamma^{xT} \mathbf{x}$ by calling the adjoint routine `aj_Gam()`:

```
call aj_Gam(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
            n_x,ktab,jtab, nvecs, ldx, x, n_x, Tx    )
```

4. Calculate $\mathtt{Ty} = \Sigma^x \mathtt{Tx}$ by calling `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
    n_x,sigW, nvecs, n_x, Tx, n_x, Ty                 )
```

5. Set $\mathtt{Tx}(1:\mathtt{n_x}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Tx} = C^x \mathtt{Ty}$. This is accomplished by calling the routine `sym_Cxpy()`:

```
call sym_Cxpy(kind_mat,kind_covV, sparse,          &
    n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW,       &
    n_x,    kx,   ks, ktab,                        &
    qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y, &
    nvecs, n_x, Ty, n_x, Tx, istat                 )
```

6. Calculate $\mathtt{Ty} = \Sigma^x \mathtt{Tx}$ by calling the routine `mv_diag()`:

```
call mv_diag(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
    n_x,sigW, nvecs, n_x, Tx, n_x, Ty                 )
```

7. Set the work array $\mathtt{Tx}(1:\mathtt{n\_x}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Tx} = \Gamma^x \mathtt{Ty}$ by calling the routine `mv_Gam()`:

```
call mv_Gam(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_lenW, &
    n_x,ktab,jtab, nvecs, n_x, Ty, n_x, Tx           )
```

The term $\Gamma^x \Sigma^x C^x \Sigma^x \Gamma^{xT} \mathbf{x}$ is stored in the array `Tx(1:n_x,1:nvecs)`, which is added to the result **y** stored in `y(1:n_x,1:nvecs)`:

```
y(1:n_x,1:nvecs) = y(1:n_x,1:nvecs) + Tx(1:n_x,1:nvecs)
```

This completes the calculation and application of the Innovation operator. The final step is to deallocate the temporary arrays `Tx` , `Ty`, `kt_lenW`, and `sigW` and return.

Table 23: Summary of arguments to `op_Mx()`.

Variable	Type/Dimensions	Intent	Description
kind_mat	INTEGER	IN	Matrix Type
kind_cov	INTEGER	IN	Covariance Type
n_kr	INTEGER	IN	Number of Regions
kr_loc	INTEGER(n_kr)	IN	Index of Region Start
kr_len	INTEGER(n_kr)	IN	Region Lengths
n_kt	INTEGER	IN	Number of Data Types
kt_loc	INTEGER(n_kt,n_kr)	IN	Start of each kt block for each region
kt_len	INTEGER(n_kt,n_kr)	IN	Length of each kt block for each region
n_x	INTEGER	IN	Dimension of Attribute Arrays
kx	INTEGER(n_x)	IN	Data Source Index
ks	INTEGER(n_x)	IN	Sounding Index
ktab	INTEGER(n_x)	IN	Level Index for Look-Up Tables
jtab	INTEGER(n_x)	IN	Latitude Index for Look-Up Tables
sigU	REAL(n_x)	IN	σ_{ou}
sigC	REAL(n_x)	IN	σ_{oc}
sigF	REAL(n_x)	IN	Forecast Error σ
qr_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$
qr_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$
qr_z	REAL(n_x)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$
qm_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$
qm_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$
qm_z	REAL(n_x)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$
ql_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$
ql_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$
nvecs	INTEGER	IN	Number of vectors
ldx	INTEGER	IN	Leading dimension of $\mathbf{x}$
x	REAL(ldx,nvecs)	IN	Input vectors $\mathbf{x}$
ldCx	INTEGER	IN	Leading dimension of Cx
Cx	REAL(ldy,nvecs)	OUT	Output vectors $(HP^fH^T + R)\mathbf{x}$

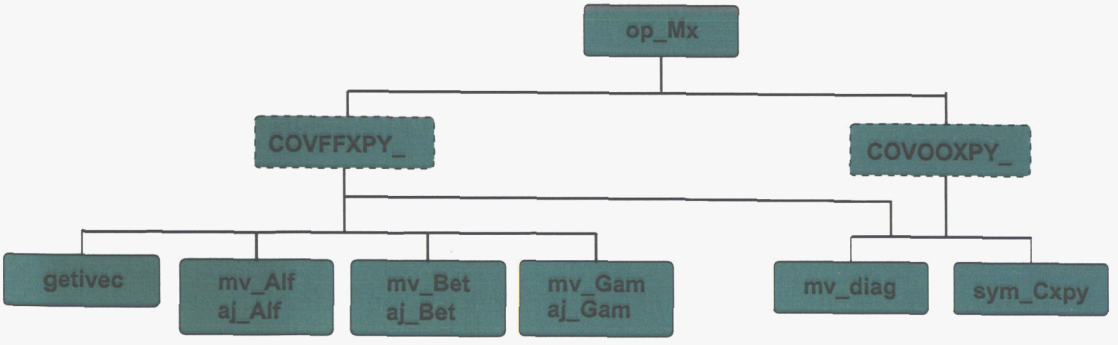


Figure 45: Calling tree for `op_Mx()`.

9 Stage III: Solution of the Analysis Equation

9.1 Calculation of Analysis Increments — getAinc()

The solution vector $\mathbf{x}$ to Eqn. (1) **xvec** is transformed to state space using Eqn. (2) to produce AI $(\mathbf{w}^a - \mathbf{w}^f)$. This is accomplished by calling the routine **getAinc()**, whose calling tree is illustrated in Figure 46. The interface to **getAinc()** is given below, and Table 24 is a summary of the routine's arguments.

```
subroutine getAinc( verbose, luverb, nbandmx, nnobs,          &
                    iregbeg, ireglen, ityplen,                &
                    xvec, rlats, rlons,                       &
                    rlevs, sigF,                              &
                    iaidim, jaidim,                           &
                    nlevai, plevai,                           &
                    usai, vsai, slpai,                        &
                    uuai, vvai, HHai, qqai,                   &
                    ussigF, vssigF, slpsigF,                  &
                    uusigF, vvsigF, HHsigF, qqsigF,           &
                    want_slu, want_slv, want_slp, want_uwnd,  &
                    want_vwnd, want_hght, want_mixr,          &
                    ierr                                       )
```

The AI are returned in the arrays **usai** (u_{sl}), **vsai** (v_{sl}), **slpai** (p_{sl}), **uuai** (u), **vvai** (v), **HHai** (h), and **qqai** (q). The input **LOGICAL** variables **want_slu**, **want_slv**, **want_slp**, **want_uwnd**, **want_vwnd**, **want_hght**, and **want_mixr** determine the variables for which AI are calculated. These variables were initialized from the resource file by the routine **iniainc()**. For example, if **want_slu** = .**TRUE.** then AI for (p_{sl}) are calculated. If **want_slu** = .**FALSE.** then AI for (p_{sl}) are not calculated.

The rows and columns of the matrix $P^f H^T$ are defined as follows:

- The columns of $P^f H^T$ are defined in observation space using the **xvec** and the observation attribute vectors.
- The rows of $P^f H^T$ are defined in state space in using **multivariate forecast attribute vectors**, which are defined below.

Multivariate forecast attribute vectors. The calculation of AI by **getAinc()** uses data from the analysis grid stored in two different types of attribute vectors: 'G-vectors' and 'V-vectors' both of which are vectors on the Quasi-Equal-Area (QEA) grid, but which store data in different order. The G-vectors are sorted to facilitate the interpolation of data back to the analysis (lat-lon) grid, i.e., data on levels are contiguous in memory. The V-vectors are sorted so that they can be proper arguments for the covariance matrices, i.e., data are sorted like $\mathbf{x}$ vectors. Specifically:

- **G-vectors**: These multivariate forecast attribute vectors are calculated in **getAinc()** and are passed into and returned from the routines **intp_sigF()** and **mvPHx()**. These vectors are sorted in lexicographic order by variable, level, region, and profile (φ, λ) (Figure 47). The ordering of the variables in the G-vector is defined by the following rules:

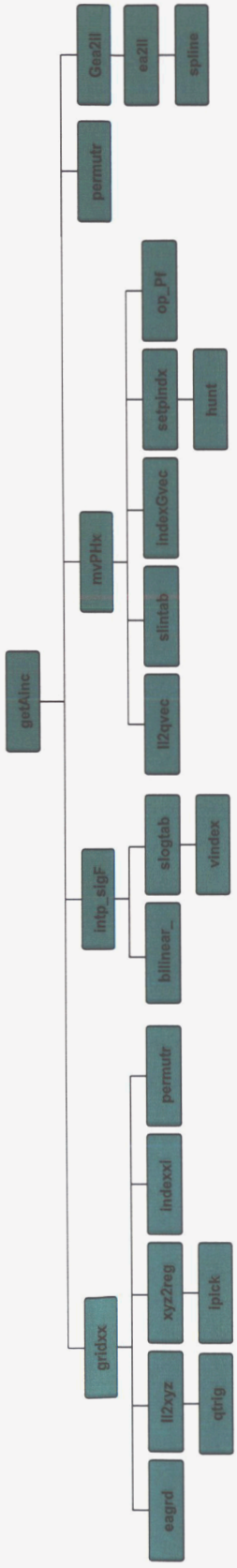


Figure 46: Calling tree for `getAinc()`.

- All the multivariate fields are packed first. The order in which they appear is: upper-air heights h, upper air zonal wind u, upper air meridional wind v, and upper-air mixing ratio q.
- Within each upper-air field, the data is organized as a series of `nlevai` levels, with their respective pressure values in descending order.
- The surface fields appear in the following order: sea-level pressure p_{sl}, sea-level zonal wind u_{sl}, sea-level meridional wind v_{sl}.

- **V-vectors**: These multivariate forecast attribute vectors are calculated from G-vectors in the routine `mvPHx()`, and passed to and returned from the routine `op_Pf()`. These vectors are sorted in lexicographic order by region, variable, level, and profile (φ, λ) (Figure 48).

The routine `getAinc()` performs the following steps:

1. Determination of the number `mxkvG` of segments in the G-vector:
   (a) Initialization: `mxkvG` = `0`
   (b) If `want_hght` = .TRUE. then `mxkvG` = `mxkvG` + `nlevai`
   (c) If `want_uwnd` = .TRUE. then `mxkvG` = `mxkvG` + `nlevai`
   (d) If `want_vwnd` = .TRUE. then `mxkvG` = `mxkvG` + `nlevai`
   (e) If `want_mixr` = .TRUE. then `mxkvG` = `mxkvG` + `nlevai`
   (f) If `want_slp` = .TRUE. then `mxkvG` = `mxkvG` + `1`
   (g) If `want_slu` = .TRUE. then `mxkvG` = `mxkvG` + `1`
   (h) If `want_slv` = .TRUE. then `mxkvG` = `mxkvG` + `1`
2. Allocation of workspace variables:
   - `INTEGER krbegG(maxreg)`: Region start index for QEA attribute vectors
   - `INTEGER krlenG(maxreg)`: Region lengths for QEA attribute vectors
   - `INTEGER kpmapG(mxkvG)`: Vertical level index for QEA attribute vectors.
   - `INTEGER ktmapG(mxkvG)`: Datatype index for QEA attribute vectors.
   - `INTEGER invperm(mxG)`: Inverse of the region sorting permutation for QEA grid locations.
   - `REAL xgrid(mxG)`: Cartesian coordinate x for QEA point on the unit sphere.
   - `REAL ygrid(mxG)`: Cartesian coordinate y for QEA point on the unit sphere.
   - `REAL zgrid(mxG)`: Cartesian coordinate z for QEA point on the unit sphere.
   - `REAL rlatG(mxG)`: QEA grid latitude values.
   - `REAL rlonG(mxG)`: QEA grid longitude values.
   - `REAL rlevG(mxG)`: QEA grid vertical level values.
   - `INTEGER ktbegG(ktmax,maxreg)`: Beginning points of region/datatype segments in QEA grid attribute vectors.
   - `INTEGER ktlenG(ktmax,maxreg)`: Lengths of region/datatype segments in QEA grid attribute vectors.
   - `REAL Gout(mxG,mxkvG)`: Analysis increment values on the QEA grid.
   - `REAL GsigF(mxG,mxkvG)`: Forecast error standard deviations σ_f on the QEA grid
3. Memory mapping of datatype and level segments of the G-vector. Once the workspace is allocated, indices for pressure levels `kpmapG(1:mxkvG)` and data type `ktmapG(1:mxkvG)` are initialized using the following rules:

- If segment **nkvG** corresponds to a surface quantity, **kpmapG(nkvG)** = 0. If segment **nkvG** corresponds to an upper-air variable, the level number **kp** $\in \{1, \ldots,$ **nlevai**$\}$ is stored in **kpmapG(nkvG)**.
- The appropriate datatype flag from the list {**ktHH**,**ktuu**,**ktvv**,**ktqq**,**ktslp**,**ktus**,**ktvs**} is stored in **ktmapG(nkvG)**.

4. The initialization of the QEA horizontal grid is performed by calling the routine **gridxx()**:

```
call gridxx ( iaidim, jaidim, nG,                        &
              krbegG, krlenG, invperm, rlatG, rlonG, &
              xgrid, ygrid, zgrid, ier                    )
```

The regular analysis grid is **idaidim** latitudes by **jdaidim** longitudes, and **nG** is the number of QEA gridpoints (**nG** < **idaidim** $*$ **jaidim**) is returned. The attributes for the horizontal QEA gridpoints are returned in vector form, sorted into regional segments as shown in Figure 6. The horizontal QEA grid attributes vectors returned by **gridxx()** are:

- Index of the beginning of each **kr**-segment of the QEA grid vector **krbegG(1:maxreg)** and its corresponding length **krlenG(1:maxreg)**
- Latitudes and longitudes of the QEA gridpoints: **rlatG(1:nG)**, and **rlonG(1:nG)**
- Cartesian coordinates of the QEA gridpoints on the unit sphere: **xgrid(1:nG)**, **ygrid(1:nG)**, and **zgrid(1:nG)**

5. Interpolation of forecast error standard deviations. This is done for each level of each field as a loop over **kv** = 1, **mxkvG**. For each value of **kv** the following steps are performed:

   (a) the **kt**-indexing arrays are initialized:

```
ktbegG(1:ktmapG(kv),1:maxreg) = 0
ktlenG(1:ktmapG(kv)-1,1:maxreg) = 0
ktlenG(1:ktmapG(kv),1:maxreg) = krlen(1:maxreg)
ktbegG(ktmapG(kv)+1,ktmax,1:maxreg) = 0
ktlenG(ktmapG(kv)+1,ktmax,1:maxreg) = krlen(1:maxreg)
```

   (b) Pressure values **rlevG** for each of the **nG** points in this **kv**-slice are assigned. For upper-air variables, **rlevG(1:nG) = plevai(kpmapG(kv))**. Surface quantities have **rlevG(1:nG) = pres4slp** (currently set to 1000 hPa).

   (c) The actual values of σ_f for this **kv** slice are calculated by interpolating input values of σ_f that are on the analysis grid and stored in the arrays **slpsigF**, **ussigF**, **vssigF**, **HHsigF**, **uusigF**, **vvsigF**, and **qqsigF**. This is implemented as a call to the routine **intp_sigF()**:

```
call intp_sigF(iaidim,jaidim,nlevai,plevai,                          &
               slpsigF,ussigF,vssigF,HHsigF,uusigF,vvsigF,qqsigF, &
               maxreg, krbegG,krlenG,ktmax,ktbegG,ktlenG,          &
               nG,rlatG,rlonG,rlevG, GsigF(1,kv))
```

   The forecast error standard deviations for a given value of **kv** are returned in the array **GsigF(1:nG,kv)**.

6. The computation of the matrix $P^f H^T$ and calculation $P^f H^T \mathbf{x}$ is accomplished by calling **mvPHx()**:

```
call mvPHx( nbandmx, maxreg, iregbeg, ireglen, ityplen, &
            nnobs, rlats, rlons, rlevs, sigF,           &
            maxreg, krbegG, krlenG, nG, rlatG, rlonG,   &
            nkvG, ktmapG, kpmapG, nlevai, plevai,       &
            mxG,   GsigF, 1, nnobs,    xvec,            &
            mxG,   mxkvG,    Gout )
```

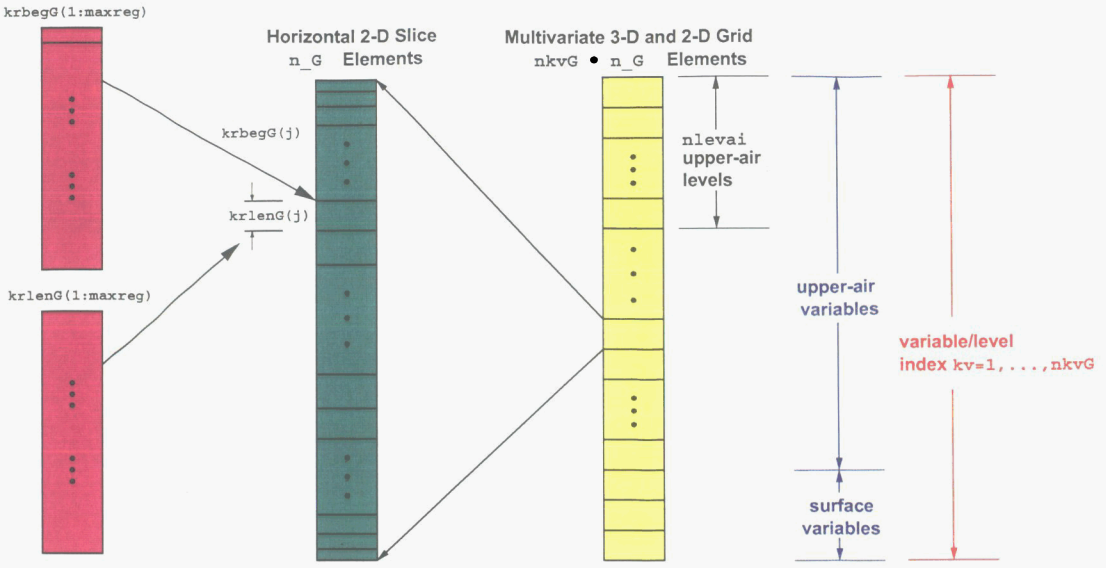


Figure 47: Organization of the QEA "G-vector" grid

The array `Gout(1:nkvG*nG)` contains the output AI on the QEA grid. A complete discussion of `mvPHx()` is presented in Section 9.2.

7. The horizontal interpolation of the analysis increments from the QEA grid back to the analysis grid is accomplished using bicubic spline interpolation. This is implemented as a call to the subroutine `Gea2ll()`:

```
call Gea2ll( nG, nkvG, ktmapG, kpmapG, 1, mxG, mxkvG, Gout, &
             iaidim, jaidim, nlevai,                        &
             HHai, uuai, vvai, qqai, slpai, usai, vsai       )
```

   The output analysis increments are in the arrays `HHai`, `uuai`, `vvai`, `qqai`, `slpai`, `usai`, and `vsai`.

8. Deallocation of dynamical workspace arrays.

9.2 Software Implementation of $P^f H^T$ — `mvPHx()`

The global matrix multiplication $P^f H^T \mathbf{x}$ is implemented in the routine `mvPHx()`:

```
subroutine mvPHx( nbandmx, n_krX, kr_locX, kr_lenX, kt_lenX,          &
                  n_X, rlatX, rlonX, rlevX, XsigF,                    &
                  n_krG, kr_locG, kr_lenG, n_G, rlatG, rlonG,         &
                  nkvG, ktmapG, kpmapG, nlevG, plevG, ldnGs, GsigF, & 
                  nvecs, ldnX, Xvec, ldnG, ldnkvG, Gout               )
```

This routine takes various inputs organized as G-vectors on the QEA grid and observation vectors, calculates $\mathbf{w}^a - \mathbf{w}^f = P^f H^T \mathbf{x}$, and returns the AI $(\mathbf{w}^a - \mathbf{w}^f)$ as a G-vector. The input G-vector is organized in the lexicographic ordering illustrated in Figure 47 and the

Table 24: Arguments to the routine `getAinc()`.

Variable	Type	Intent	Description
verbose	LOGICAL	IN	Verbosity Control Flag
luverb	INTEGER	IN	Diagnostic Output Device
nbandmx	INTEGER	IN	Number of Matrix Bands
nnobs	INTEGER	IN	Number of Observations
iregbeg	INTEGER(maxreg)	IN	Region Start Index
ireglen	INTEGER(maxreg)	IN	Region Length
ityplen	INTEGER(ktmax,maxreg)	IN	Region/Datatype Segment Length
xvec	REAL(nnobs)	IN	Intermediate vector $\mathbf{x}$
rlats	REAL(nnobs)	IN	Latitude
rlons	REAL(nnobs)	IN	Longitude
rlevs	REAL(nnobs)	IN	Pressure (hPa)
sigF	REAL(nnobs)	IN	σ_f
iaidim	INTEGER	IN	Number Analysis Latitudes
jaidim	INTEGER	IN	Number Analysis Longitudes
nlevai	INTEGER	IN	Number of Analysis Levels
plevai	REAL(nlevai)	IN	Analysis Levels
usai	REAL(iaidim,jaidim)	OUT	Sea-level u AI's
vsai	REAL(iaidim,jaidim)	OUT	Sea-level v AI's
slpai	REAL(iaidim,jaidim)	OUT	Sea-level Pressure AI's
uuai	REAL(iaidim,jaidim,nlevai)	OUT	Upper-air u AI's
vvai	REAL(iaidim,jaidim,nlevai)	OUT	Upper-air v AI's
HHai	REAL(iaidim,jaidim,nlevai)	OUT	Upper-air h AI's
qqai	REAL(iaidim,jaidim,nlevai)	OUT	Upper-air q AI's
ussigF	REAL(iaidim,jaidim)	IN	σ_f for Sea-level u
vssigF	REAL(iaidim,jaidim)	IN	σ_f for Sea-level v
slpsigF	REAL(iaidim,jaidim)	IN	σ_f for Sea-level Pressure
uusigF	REAL(iaidim,jaidim,nlevai)	IN	σ_f for Upper-air u
vvsigF	REAL(iaidim,jaidim,nlevai)	IN	σ_f for Upper-air v
HHsigF	REAL(iaidim,jaidim,nlevai)	IN	σ_f for Upper-air h
qqsigF	REAL(iaidim,jaidim,nlevai)	IN	σ_f for Upper-air q
want_slu	LOGICAL	IN	Want u_{sl} AI's
want_slv	LOGICAL	IN	Want v_{sl} AI's
want_slp	LOGICAL	IN	Want Sea-level Pressure AI's
want_uwnd	LOGICAL	IN	Want Upper-air u AI's
want_vwnd	LOGICAL	IN	Want Upper-air v AI's
want_hght	LOGICAL	IN	Want Upper-air h AI's
want_mixr	LOGICAL	IN	Want Upper-air q AI's
ierr	INTEGER	OUT	Error Flag

input vector **x** is sorted in lexicographic order, as illustrated in Figure 5. Block structure is imposed in $P^f H^T$ by organizing its elements in region-region blocks, which requires the input G-vectors to be transformed into V-vectors (Figure 48). Once the forecast error standard deviations and additional attributes necessary to calculate $P^f H^T \mathbf{x}$ are stored in V-vector form, the actual matrix-vector multiplication is done by calling `op_Pf()`. The result from `op_Pf()` is AI in V-vector form, which are transformed back to G-vector form, completing the calculation.

The routine `mvPHx()` performs the following steps:

1. The argument checks listed below:
   (a) The number of innovations `n_X` is less than or equal to `ldnX`, the leading dimension of `Xvec`.
   (b) The number elements in a QEA 2-d slice `n_G` is less than or equal to `ldnG`, the first leading dimension of the output G-vector `Gout`.
   (c) The number `kt`/level segments `nkvG` is less than or equal to `ldnkvG`, the second leading dimension of the output G-vector `Gout`.

   If any of these conditions are violated, a diagnostic message is written to `stderr`, and execution is terminated via a call to `psasexit()`.

2. Allocation of temporary workspace variables listed below:
   - `kr_locV(n_krG)` and `kr_lenV(n_krG)`: Region index and lengths for V-vectors.
   - `kt_locV(ktmax,n_krG)` and `kt_lenV(ktmax,n_krG)`: Index and lengths for the `kr`/`kt` blocks for V-vectors.
   - `kv_locV(nkvG,n_krG)` and `kt_lenV(nkvG,n_krG)`: Index and lengths of `kr`/`kv` segments.
   - `qrV_x(n_V)`, `qrV_y(n_V)`, `qrV_z(n_V)`, `qmV_x(n_V)`, `qmV_y(n_V)`, `qmV_z(n_V)`, `qlV_x(n_V)`, `qlV_y(n_V)`: Cartesian components of $\hat{\mathbf{e}}_r$, $\hat{\mathbf{e}}_l$, and $\hat{\mathbf{e}}_m$ organized as V-vectors. The quantity `n_V` is the total number of forecast values in the V-vector; `n_V` = `n_G` $*$ `nkvG`.
   - `qrG_x(n_G)`, `qrG_y(n_G)`, `qrG_z(n_G)`, `qmG_x(n_G)`, `qmG_y(n_G)`, `qmG_z(n_G)`, `qlG_x(n_G)`, `qlG_y(n_G)`: Cartesian components of $\hat{\mathbf{e}}_r$, $\hat{\mathbf{e}}_l$, and $\hat{\mathbf{e}}_m$ organized as G-vectors.
   - `ktabV(n_V)` and `jtabV(n_V)`: level and latitude indices for imat structures, organized as V-vectors.
   - `jtabG(n_G)` and `vtabG(n_G)`: latitude index and interpolation weight for imat structures, organized as G-vectors.
   - `kt_locX(ktmax,n_krX)`: the `kt`-index array for the input vector `Xvec`.
   - `qrX_x(n_X)`, `qrX_y(n_X)`, `qrX_z(n_X)`, `qmX_x(n_X)`, `qmX_y(n_X)`, `qmX_z(n_X)`, `qlX_x(n_X)`, `qlX_y(n_X)`: Cartesian components of $\hat{\mathbf{e}}_r$, $\hat{\mathbf{e}}_l$, and $\hat{\mathbf{e}}_m$ at observation locations.
   - `jtabX(n_X)` and `vtabX(n_X)`: latitude index and interpolation weights for observation locations.
   - `ktabX(n_X)` and `wtabX(n_X)`: vertical level index and interpolation weights for observation locations
   - `VsigF(n_V)`: Forecast error standard deviations in V-vector form.
   - `Vout(n_V,nvecs)`: V-vector output result $P^f H^T \mathbf{x}$ returned from `op_Pf()`.

3. Calculation of the Cartesian components of the unit vectors $\hat{\mathbf{e}}_r$, $\hat{\mathbf{e}}_l$, and $\hat{\mathbf{e}}_m$ for the horizontal QEA grid points are calculated by calling the routine `ll2qvec()`:

```
call ll2qvec(n_G,rlatG,rlonG,qrG_x,qrG_y,qrG_z, &
             qmG_x,qmG_y,qmG_z,qlG_x,qlG_y       )
```

4. Initialization of IMAT latitudinal index and weight arrays for the G-vector by calling the routine **slintab()**:

```
call slintab(roundoff,nveclat,veclats,n_G,rlatG,jtabG,vtabG)
```

The imat latitude index is returned in **jtabG(1:n_G)** and the corresponding linear interpolation weight in **vtabG(1:n_G)**.

5. Initialize the memory mapping between the G-vector and V-vector representations of the forecast attribute vectors. This information is contained in the arrays **kr_locV**, **kr_lenV**, **kv_locV**, **kv_lenV**, **kt_locV**, and **kt_lenV**, which are set by a call to the routine **indexGvec()**:

```
call indexGvec(n_krG, kr_locG, kr_lenG, nkvG, ktmapG,            &
               n_krG, kr_locV, kr_lenV, ktmax, kt_locV, kt_lenV, &
               nkvG, kv_locV, kv_lenV                             )
```

6. Mapping of the G-vector attributes **qrG_x**, **qrG_y**, **qrG_z**, **qmG_x**, **qmG_y**, **qmG_z**, **qlG_x**, **qlG_y**, **jtabG**, and **vtabG** into their respective V-vector attributes **qrV_y**, **qrV_z**, **qmV_x**, **qmV_y**, **qmV_z**, **qlV_x**, **qlV_y**, **jtabV**, and **VsigF**. This is done by the following block of code:

```
do kr=1,n_krG
  lcG=kr_locG(kr)
  lnG=kr_lenG(kr)
  leG=lcG+lnG-1

  do kv=1,nkvG
    lc=kr_locV(kr)+kv_locV(kv,kr)
    ln=kv_lenV(kv,kr)
    le=lc+ln-1

    VsigF(lc:le)=GsigF(lcG:leG,kv)

    qrV_x(lc:le)=qrG_x(lcG:leG)
    qrV_y(lc:le)=qrG_y(lcG:leG)
    qrV_z(lc:le)=qrG_z(lcG:leG)
    qmV_x(lc:le)=qmG_x(lcG:leG)
    qmV_y(lc:le)=qmG_y(lcG:leG)
    qmV_z(lc:le)=qmG_z(lcG:leG)
    qlV_x(lc:le)=qlG_x(lcG:leG)
    qlV_y(lc:le)=qlG_y(lcG:leG)

    jtabV(lc:le)=jtabG(lcG:leG)

  end do
end do
```

7. The vertical level imat indices and weights are determined by calls to the routine **setpindx()**. This is done one value of **kv** at a time. For upper-air fields, the local pressure level value is **kpmapG(kv)**, and the call takes this form:

```
call setpindx(1,plevG(kpmapG(kv)),wlev,nveclev,pveclev)
```

For surface variables, the sea-level pressure value is taken to be 1000 hPa, and **setpindx()** is called as follows:

```
call setpindx(1,1000.,wlev,nveclev,pveclev)
```

In both cases, the nearest pressure value is returned in the **REAL** variable **wlev**. This value is converted to the appropriate **INTEGER** index **klev** as follows:

```
klev=min(int(wlev),nveclev-1)
wlev=wlev-klev
klev=klev + nint(wlev)
```

and the appropriate values of klev are stored in the array **ktabV**.

8. Calculation of the observation grid attributes **qrX_x**, **qrX_y**, **qrX_z**, **qmX_x**, **qmX_y**, **qmX_z**, **qlX_x**, **qlX_y**, **jtabX**, and **ktabX**. The Cartesian components of the unit vectors $\hat{\mathbf{e}}_r$, $\hat{\mathbf{e}}_l$, and $\hat{\mathbf{e}}_m$ are calculated using a call to **ll2qvec()**:

```
call ll2qvec(n_X,rlatX,rlonX,qrX_x,qrX_y,qrX_z, &
             qmX_x,qmX_y,qmX_z,qlX_x,qlX_y       )
```

9. The observation IMAT latitude table indices **jtabX** are calculated by calling **slintab()**:

```
call slintab(roundoff,nveclat,veclats,n_X,rlatX,jtabX,vtabX)
```

10. The observation IMAT level table indices **ktabX** are calculated by calling **slogtab()**:

```
call slogtab(roundoff,nveclev,pveclev,n_X,rlevX,ktabX,wtabX)
```

11. The computation $P^f H^T \mathbf{x}$ is performed in this forecast V-vector/Observation Vector representation by calling the routine **op_Pf()**:

```
call op_Pf(nbandmx,kind_covF.or.kind_covS.or.kind_covV,     &
           n_krG, kr_locV,kr_lenV, ktmax, kt_locV,kt_lenV,  &
           n_V, ktabV,jtabV,VsigF,                          &
           qrV_x,qrV_y,qrV_z,qmV_x,qmV_y,qmV_z,qlV_x,qlV_y, &
           n_krX, kr_locX,kr_lenX, ktmax, kt_locX,kt_lenX,  &
           n_X, ktabX,jtabX,XsigF,                          &
           qrX_x,qrX_y,qrX_z,qmX_x,qmX_y,qmX_z,qlX_x,qlX_y, &
           nvecs, ldnX, Xvec, n_V, Vout(1,1)                 )
```

The AI in V-vector form is returned in **Vout(1:n_V,1:nvecs)**.

12. The V-vector AI **Vout** is then mapped to its G-vector representation **Gout**:

```
do kr=1,n_krG
  lcG=kr_locG(kr)
  lnG=kr_lenG(kr)
  leG=lcG+lnG-1

  do kv=1,nkvG
    lc=kr_locV(kr)+kv_locV(kv,kr)
    ln=kv_lenV(kv,kr)
    le=lc+ln-1

    Gout(lcG:leG,kv,1:nvecs) = Vout(lc:le,1:nvecs)

  end do
end do
```

13. The temporary arrays allocated in this routine are deallocated.

The routine **mvPHx()** returns the AI in G-vector form as **Gout(lcG:leG,kv,1:nvecs)**.

Table 25: Arguments to the routine `mvPHx()`.

Variable	Type	Intent	Description
nbandmx	INTEGER	IN	Number of bands in P^f
n_krX	INTEGER	IN	Number of Observation Regions
kr_locX	INTEGER(n_krX)	IN	Region Start Index (Obs)
kr_lenX	INTEGER(n_krX)	IN	Region Length (Obs)
kt_lenX	INTEGER(ktmax,n_krX)	IN	Region/Datatype Segment Length (Obs)
n_X	INTEGER	IN	Intermediate vector $\mathbf{x}$
rlatX	REAL(n_X)	IN	Latitude (Obs)
rlonX	REAL(n_X)	IN	Longitude (Obs)
rlevX	REAL(n_X)	IN	Pressure (hPa)
XsigF	REAL(n_X)	IN	σ_f (obs space)
n_krG	INTEGER	IN	Number of Forecast Regions
kr_locG	INTEGER(n_krG)	IN	Region Start Index (Fcast)
kr_lenG	INTEGER(n_krG)	IN	Region Length (Fcast)
kt_lenG	INTEGER(ktmax,n_krG)	IN	Region/Datatype Segment Length (Fcast)
n_G	INTEGER	IN	# of horizontal QEA grid points
rlatG	REAL(n_G)	IN	Horizontal QEA latitudes
rlonG	REAL(n_G)	IN	Horizontal QEA longitudes
nkvG	INTEGER	IN	Number of **kv**-slices
ktmapG	INTEGER(nkvG)	IN	**kt** values for **kv**-slices
kpmapG	INTEGER(nkvG)	IN	Pressure values for **kv**-slices
nlevG	INTEGER	IN	Number of Analysis pressure levels
plevG	REAL(nlevG)	IN	Analysis pressure levels (hPa)
ldnGs	INTEGER	IN	Length of `GsigF`
GsigF	REAL(ldnGs)	IN	G-vector forecast error
nvecs	INTEGER	IN	Number of $\mathbf{x}$ vectors
ldnX	INTEGER	IN	Leading dimension of `Xvec`
Xvec	REAL(ldnX,nvecs)	IN	$\mathbf{x}$
ldnG	INTEGER	IN	First leading dimension of `Gout`
ldnkvG	INTEGER	IN	Number of **kv**-slices in `Gout`
Gout	REAL(ldnG,ldnkvG,nvecs)	OUT	G-vector $P^f H^T \mathbf{x}$

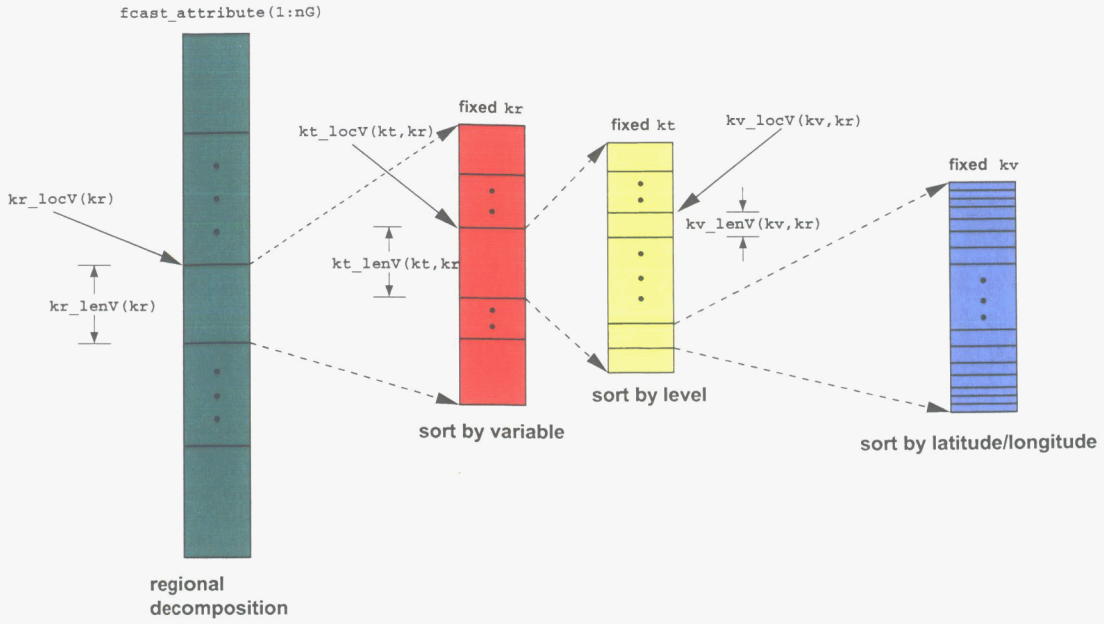


Figure 48: Organization of the "V-vector" grid

.

9.2.1 V-vector Calculation of $P^f H^T \mathbf{x}$ — op_Pf()

The calculation of $P^f H^T \mathbf{x}$ in the forecast V-vector/observation grid representation occurs in subroutine op_Pf():

```
subroutine op_Pf(kind_mat,kind_cov,                                    &
                 n_kri,kri_loc,kri_len, n_kti,kti_loc,kti_len,         &
                 n_i,ktabi,jtabi,sigFi,                                &
                 qri_x,qri_y,qri_z,qmi_x,qmi_y,qmi_z,qli_x,qli_y,      &
                 n_krj,krj_loc,krj_len, n_ktj,ktj_loc,ktj_len,         &
                 n_j,ktabj,jtabj,sigFj,                                &
                 qrj_x,qrj_y,qrj_z,qmj_x,qmj_y,qmj_z,qlj_x,qlj_y,      &
                 nvecs, ldxj, xj, ldCxj, Cxj                            )
```

The arguments to op_Pf() are summarized in Table 9.2.1 and its calling tree is illustrated in Figure 49.

Execution begins with a check of the argument kind_cov to assure that it is one of the following: kind_covF, kind_covS, or kind_covF. If it is not one of these values, kind_cov does not correspond to any kind of forecast error covariance operator, and op_Pf() writes an error message to stderr and terminates via a call to psasexit().

The result V-vector array Cxj is initialized:

```
Cxj(1:n_i,1:nvecs) = 0.
```

The calculation of the matrix-vector product Cxj is performed by an internal procedure covFFxpy_:

```
call covFFxpy_(ldxj, xj, ldCxj, Cxj)
```

The workspace used is dynamically allocated and includes:

- `Tx(ldT,nvecs)` and `Ty(ldT,nvecs)`: intermediate result arrays.
- `kti_lenW(n_kti,n_kri)` and `ktj_lenW(n_ktj,n_krj)`: `kt/kr` block dimensions for the rows (V-vector) and columns (obs. vector).
- `sigWi(ldT)` and `sigWj(ldT)`: Vectors used to store forecast σ^{ψ} and σ^{χ} values.

The `INTEGER` parameter `ldT` is the maximum of `ldxj` and `ldyi`.

The calculation of $P^f H^T \mathbf{x}$ is similar to the calculation of $(HP^f H^T + R)\mathbf{x}$ described in Section 8.3.

The first step in `covFFxpy_()` is to initialize `Tx(1:n_j1:nvecs)` to zero. The term $\Gamma^h \Sigma^h C^h \Sigma^h \Gamma^{hT} \mathbf{x}$ is calculated using the following steps:

1. Initialize the temporary array `Tx(1:n_x,1:nvecs)` to zero.
2. Calculate `Tx` $= \Gamma^{hT}\mathbf{x}$ by calling the routine `aj_Alf()`:

```
call aj_Alf(n_krj,krj_loc,krj_len, n_ktj,ktj_loc,ktj_len, &
            n_j,ktabj,jtabj, nvecs, ldxj, xj, ldT, Tx      )
```

3. Calculate `Ty` $= \Sigma^h$`Tx` by calling `mv_diag()`:

```
call mv_diag(n_krj,krj_loc,krj_len,n_ktj,ktj_loc,ktj_len, &
             n_j, sigFj, nvecs, ldT, Tx, ldT, Ty           )
```

4. Set `Tx`$(1:$`n_i`$, 1:$`nvecs`$) = 0$. Calculate `Tx` $= C^h$`Ty` by calling the routine `rec_Cxpy()`:

```
call rec_Cxpy(kind_mat,kind_covF, sparse,                      &
              n_kri,kri_loc,kri_len, n_kti,kti_loc,kti_len,    &
              n_i,   ktabi,                                    &
              qri_x,qri_y,qri_z,qmi_x,qmi_y,qmi_z,qli_x,qli_y, &
              n_krj,krj_loc,krj_len, n_ktj,ktj_loc,ktj_len,    &
              n_j,   ktabj,                                    &
              qrj_x,qrj_y,qrj_z,qmj_x,qmj_y,qmj_z,qlj_x,qlj_y, &
              nvecs, ldT, Ty, ldT, Tx, istat                    )
```

5. Calculate `Ty` $= \Sigma^h$`Tx` by calling the routine `mv_diag()`:

```
call mv_diag(n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_len, &
             n_i,sigFi, nvecs, ldT, Tx, ldT, Ty           )
```

6. Set the work array `Tx`$(1:$`n_i`$, 1:$`nvecs`$) = 0$. Calculate `Tx` $= \Gamma^h$`Ty` by calling the routine `mv_Alf()`:

```
call mv_Alf(n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_len, &
            n_i,ktabi,jtabi, nvecs, ldT, Ty, n_x, Tx      )
```

The term $\Gamma^h \Sigma^h C^h \Sigma^h \Gamma^{hT} \mathbf{x}$ is stored in `Tx(1:n_i,1:nvecs)`, and added to the result $\mathbf{y}$ stored in `yi(1:n_i,1:nvecs)`:

```
yi(1:n_i,1:nvecs) = yi(1:n_i,1:nvecs) + Tx(1:n_i,1:nvecs)
```

The mass-decoupled forecast wind error covariance terms are calculated in a manner similar to the mass-coupled forecast wind error covariance terms. The lengths of the segments of $\mathbf{x}$ that contain variables that are either upper-air or surface wind components are stored in a dynamically allocated arrays `kti_lenW(1:n_kti,1:n_kri)` and `ktj_lenW(1:n_ktj,1:n_krj)`. The forecast error standard deviations σ^ψ and σ^χ are stored in `sigWi(1:n_i)` and `sigWj(1:n_j)`. The windfield segment indexing information is set in `kt_lenWi` and `kt_lenWj` as follows:

```
kti_lenW(:,:)=0
kti_lenW(ktus,:)=kti_len(ktus,:)
kti_lenW(ktvs,:)=kti_len(ktvs,:)
kti_lenW(ktuu,:)=kti_len(ktuu,:)
kti_lenW(ktvv,:)=kti_len(ktvv,:)

ktj_lenW(:,:)=0
ktj_lenW(ktus,:)=ktj_len(ktus,:)
ktj_lenW(ktvs,:)=ktj_len(ktvs,:)
ktj_lenW(ktuu,:)=ktj_len(ktuu,:)
ktj_lenW(ktvv,:)=ktj_len(ktvv,:)
```

The vector $\Gamma^\psi \Sigma^\psi C^h \Sigma^\psi \Gamma^{\psi T} \mathbf{x}$ is calculated for both the upper-air and surface analyses as follows:

1. Initialize the wind error streamfunction standard deviation temporary arrays `sigWi(1 : n_i) = 0` and `sigWj(1 : n_j) = 0`.

2. Fill in the necessary elements of `sigWi` and `sigWj` with their respective values of σ^ψ obtained from the IMAT structure `FEsigS_imat`. This is implemented as a pair of calls to the routine `getivec()`.

```
call getivec(MXveclev,MXveclat,FEsigS_imat,                &
             n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW, &
             n_i, ktabi, jtabi, sigWi                       )
call getivec(MXveclev,MXveclat,FEsigS_imat,           &
             n_krj,krj_loc,krj_len,n_ktj,ktj_loc,ktj_lenW, &
             n_j, ktabj, jtabj, sigWj                     ) .
```

3. Set `Tx(1 : n_j, 1 : nvecs) = 0`.

4. Calculate `Tx` $= \Gamma^{\psi T} \mathbf{x}$ by calling the routine `aj_Bet()`:

```
call aj_Bet(n_krj,krj_loc,krj_len,n_ktj,ktj_loc,ktj_lenW, &
            n_j,ktabj,jtabj,nvecs,ldxj,xj,ldT,Tx           )
```

5. Calculate `Ty` $= \Sigma^\psi$`Tx` by calling `mv_diag()`:

```
call mv_diag(n_krj,krj_loc,krj_len,n_ktj,ktj_loc,ktj_lenW, &
             n_j,sigWj,nvecs,ldT,Tx,ldT,Ty                  )
```

6. Set $\mathtt{Tx}(1:\mathtt{n_i}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Tx} = C^{\psi}\mathtt{Ty}$. This is accomplished by calling the rectangular operator routine `rec_Cxpy()`:

```
call rec_Cxpy(kind_mat,kind_covF, sparse,                    &
    n_kri,kri_loc,kri_len, n_kti,kti_loc,kti_len,            &
    n_i,   ktabi,                                            &
         qri_x,qri_y,qri_z,qmi_x,qmi_y,qmi_z,qli_x,qli_y,    &
    n_krj,krj_loc,krj_len, n_ktj,ktj_loc,ktj_len,            &
    n_j,   ktabj,                                            &
         qrj_x,qrj_y,qrj_z,qmj_x,qmj_y,qmj_z,qlj_x,qlj_y,    &
  nvecs, ldT, Ty, ldT, Tx, istat                             )
```

7. Calculate $\mathtt{Ty} = \Sigma^{\psi}\mathtt{Tx}$ by calling the routine `mv_diag()`:

```
call mv_diag(n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW, &
             n_i,sigWi,nvecs,ldT,Tx,ldT,Ty                 )
```

8. Set the work array $\mathtt{Tx}(1:\mathtt{n\_i}, 1:\mathtt{nvecs}) = 0$. Calculate $\mathtt{Tx} = \Gamma^{\psi}\mathtt{Ty}$ by calling the routine `mv_Bet()`:

```
call mv_Bet(n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW, &
            n_i,ktabi,jtabi,nvecs,ldT,Ty,ldT,Tx           )
```

The term $\Gamma^{\psi}\Sigma^{\psi}C^{h}\Sigma^{\psi}\Gamma^{\psi T}\mathbf{x}$ is stored in `Tx(1:n_i,1:nvecs)`, and added to the result $\mathbf{y}$ stored in `y(1:n_i,1:nvecs)`:

```
y(1:n_i,1:nvecs) = y(1:n_i,1:nvecs) + Tx(1:n_i,1:nvecs)
```

The final error covariance operator term $\Gamma^{\chi}\Sigma^{\chi}C^{\chi}\Sigma^{\chi}\Gamma^{\chi T}\mathbf{x}$ is calculated for both the upper-air and surface analyses as follows:

1. Initialize the wind error streamfunction standard deviation temporary arrays $\mathtt{sigWi}(1:\mathtt{n\_i}) = 0$ and $\mathtt{sigWj}(1:\mathtt{n\_j}) = 0$.

2. Fill in the necessary elements of `sigWi` and `sigWj` with their respective values of σ^{χ} obtained from the IMAT structure `FEsigS_imat`. This is implemented as a pair of calls to the routine `getivec()`.

```
call getivec(MXveclev,MXveclat,FEsigV_imat,                &
             n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW, &
             n_i, ktabi, jtabi, sigWi                       )
call getivec(MXveclev,MXveclat,FEsigV_imat,                &
             n_krj,krj_loc,krj_len,n_ktj,ktj_loc,ktj_lenW, &
             n_j, ktabj,jtabj,sigWj                         ) .
```

3. Set $\mathtt{Tx}(1:\mathtt{n\_j}, 1:\mathtt{nvecs}) = 0$.

4. Calculate $\mathtt{Tx} = \Gamma^{\chi T}\mathbf{x}$ by calling the adjoint routine `aj_Gam()`:

```
call aj_Gam(n_krj,krj_loc,krj_len,n_ktj,ktj_loc,ktj_lenW, &
            n_j,ktabj,jtabj,nvecs,ldxj,xj,ldT,Tx           ) .
```

5. Calculate $\mathtt{Ty} = \Sigma^{\chi}\mathtt{Tx}$ by calling `mv_diag()`:

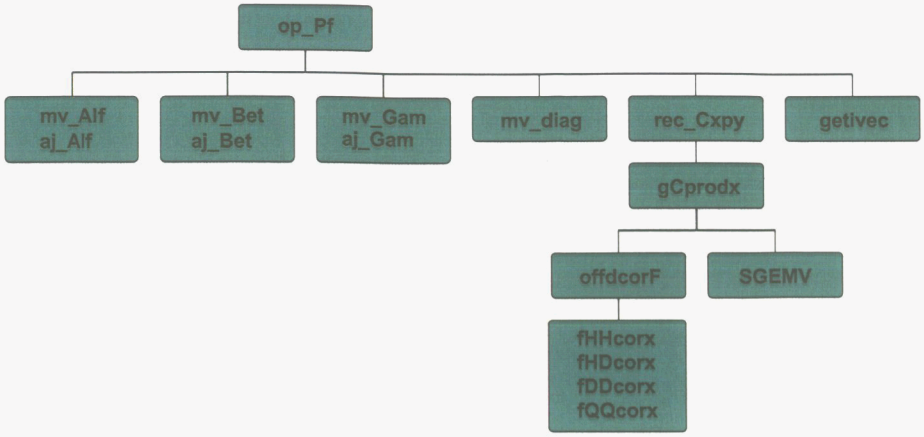


Figure 49: Calling tree for `op_Pf()`.

```
call mv_diag(n_krj,krj_loc,krj_len,n_ktj,ktj_loc,ktj_lenW, &
             n_j,sigWj,nvecs,ldT,Tx,ldT,Ty                  ) .
```

6. Set $\texttt{Tx}(1:\texttt{n}_\texttt{i}, 1:\texttt{nvecs}) = 0$. Calculate $\texttt{Tx} = C^x\texttt{Ty}$. This is accomplished by calling the rectangular operator routine `rec_Cxpy()`:

```
call rec_Cxpy(kind_mat,kind_covV,sparse,                        &
              n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW,     &
              n_i,ktabi,qri_x,qri_y,qri_z,qmi_x,qmi_y,qmi_z,    &
              qli_x,qli_y,n_krj,krj_loc,krj_len,n_ktj,ktj_loc,  &
              ktj_lenW,n_j,ktabj,qrj_x,qrj_y,qrj_z,             &
              qmj_x,qmj_y,qmj_z,qlj_x,qlj_y,                    &
              nvecs,ldT,Ty,ldT,Tx,istat                          ) .
```

7. Calculate $\texttt{Ty} = \Sigma^x\texttt{Tx}$ by calling the routine `mv_diag()`:

```
call mv_diag(n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW, &
             n_i,sigWi,nvecs,ldT,Tx,ldT,Ty                  ) .
```

8. Set the work array $\texttt{Tx}(1:\texttt{n\_i}, 1:\texttt{nvecs}) = 0$. Calculate $\texttt{Tx} = \Gamma^x\texttt{Ty}$ by calling the routine `mv_Gam()`:

```
call mv_Gam(n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW, &
            n_i,ktabi,jtabi,nvecs,ldT,Ty,ldT,Tx            ) .
```

The term $\Gamma^x\Sigma^x C^x\Sigma^x\Gamma^{xT}\mathbf{x}$ is stored in `Tx(1:n_i,1:nvecs)`, and added to the result **y** stored in `y(1:n_i,1:nvecs)`:

```
y(1:n_i,1:nvecs) = y(1:n_i,1:nvecs) + Tx(1:n_i,1:nvecs) .
```

This completes the calculation of $P^f H^T\mathbf{x}$. The temporary arrays `Tx`, `Ty`, `kti_lenW`, `ktj_lenW`, `sigWi`, and `sigWj` are deallocated, and `covFFxpy_` returns the result `y(1:n_i,1:nvecs)` to `op_Pf()`. In `op_Pf()`, the result $P^f H^T\mathbf{x}$ is returned via the interface of `covFFxpy_` to the space `Cxj(1:ldCxj,1:nvecs)`, and `op_Pf()` returns.

Table 26: Summary of arguments to `op_Pf()`.

Variable	Type/Dimensions	Intent	Description
kind_mat	INTEGER	IN	Matrix Type
kind_cov	INTEGER	IN	Covariance Type
n_kri	INTEGER	IN	Number of Regions
kri_loc	INTEGER(n_kr)	IN	Index of Region Start
kri_len	INTEGER(n_kr)	IN	Region Lengths
n_kti	INTEGER	IN	Number of Data Types
kti_loc	INTEGER(n_kt,n_kr)	IN	Start of each kt block for each region
kti_len	INTEGER(n_kt,n_kr)	IN	Length of each kt block for each region
n_i	INTEGER	IN	Dimension of Attribute Arrays
ktabi	INTEGER(n_x)	IN	Level Index for Look-Up Tables
jtabi	INTEGER(n_x)	IN	Latitude Index for Look-Up Tables
sigFi	REAL(n_x)	IN	σ_f
qri_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$
qri_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$
qri_z	REAL(n_x)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$
qmi_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$
qmi_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$
qmi_z	REAL(n_x)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$
qli_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$
qli_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$
n_krj	INTEGER	IN	Number of Regions
krj_loc	INTEGER(n_kr)	IN	Index of Region Start
krj_len	INTEGER(n_kr)	IN	Region Lengths
n_ktj	INTEGER	IN	Number of Data Types
ktj_loc	INTEGER(n_kt,n_kr)	IN	Start of each kt block for each region
ktj_len	INTEGER(n_kt,n_kr)	IN	Length of each kt block for each region
n_j	INTEGER	IN	Dimension of Attribute Arrays
ktabj	INTEGER(n_j)	IN	Level Index for Look-Up Tables
jtabj	INTEGER(n_j)	IN	Latitude Index for Look-Up Tables
sigFj	REAL(n_j)	IN	σ_f
qrj_x	REAL(n_j)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$
qrj_y	REAL(n_j)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$
qrj_z	REAL(n_j)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$
qmj_x	REAL(n_j)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$
qmj_y	REAL(n_j)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$
qmj_z	REAL(n_j)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$
qlj_x	REAL(n_j)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$
qlj_y	REAL(n_j)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$
nvecs	INTEGER	IN	Number of vectors
ldxj	INTEGER	IN	Leading dimension of $\mathbf{x}$
xj	REAL(ldxj,nvecs)	IN	Input vectors $\mathbf{x}$
ldCxj	INTEGER	IN	Leading dimension of Cx
Cxj	REAL(ldCxj,nvecs)	OUT	Output vectors $M\mathbf{x}$

10 Stage III: Software Implementation the "Lower-Level" Operators in PSAS

10.1 Mass-coupled Height/Wind Forecast Error Covariances Γ^h

The multivariate height/wind error covariance models used in the PSAS are described in [Guo et al., 1998] Sec. 4.

The multivariate operator Γ^h and its transpose Γ^{h^T} are implemented in the routines `mv_alf()` and `aj_alf()`, respectively.

```
subroutine mv_Alf( n_kr,kr_loc,kr_len,          &
                   n_kt,kt_loc,kt_len,          &
        nsize,  ktab,   jtab,                   &
        nvecs,   ldH,   Hvec,                   &
                 ldX,   Xvec                    )
```

An analytic description of the input vector `Hvec` and output vector `Xvec` appears in [Staff, 1996] Sec. 5.2.7.2.2.1, cf. [Guo et al., 1998] Sec. 4.

The routine `mv_Alf()` performs the following steps:

1. Checks that input data satisfies:
   (a) `nsize > ldH` and `nsize > ldX`.
   (b) Surface wind variable blocks of unequal length, i.e., `kt_len(ktus,kr)` $\neq$ `kt_len(ktvs,kr)` for any region index `kr`.
   (c) Upper-air wind variable blocks of unequal length, i.e., `kt_len(ktuu,kr)` $\neq$ `kt_len(ktvv,kr)` for any region index `kr`.

   The routine `mv_Alf()` exits if any of these conditions are violated.
2. Allocation of local workspace variables:
   (a) The tunable α_{um_k} and α_{ul_k} are stored in the array `a_u(1:nsize)` and the coefficients α_{vm_k} and α_{vl_k} are stored in the array `a_v(1:nsize)`. The parameters α_{um_k}, α_{ul_k}, α_{vm_k}, and α_{vl_k} are defined analytically in [Guo et al., 1998] Sec. 4.
   (b) An integer array `list_row(1:n_kt*n_kr)` contains indices of the starting points of distinct segments of the input array `Hvec`.
3. The index structure `list_row` is filled in by looping over each `kr/kt` combination, counting the number of distinct `kr/kt` segments `nrow`, and filling the entry `list_row(1:nrow)` with the index of the first element in the segment. As this index is constructed, the pairs of (u, v) segments are tested for proper alignment. The routine `mv_Alf()` exits if alignment problems are detected.
4. The calculation `Xvec` $= \Gamma^h$`Hvec` is performed in the loop described below over the index `ir` $\in$ {`1,...nrow`}:
   (a) For each value of `ir`, the value of `kr` and `kt` are determined for this `kr/kt` segment:

```
irow = list_row(ir)
kr = (irow - 1) / n_kt + 1
kt = mod(irow - 1, n_kt) + 1.
```

(b) For non-wind values of **kt**, Γ^h is the identity operator. The following steps are taken to calculate Γ^h**Hvec** for these **kr/kt** segments:

i. The beginning and end indices for this segment, `lc` and `le`, respectively, are determined:

```
lc = kr_loc(kr) + kt_loc(kt,kr)
le = lc + kt_len(kt,kr) - 1.
```

ii. The identity operator is applied to this segment of each input vector in **Hvec**:

```
Xvec(lc:le,1:nvecs) = Hvec(lc:le,1:nvecs).
```

(c) For wind values of **kt**, the calculation of Γ^h**Hvec** for the **kr/kt** segment proceeds as follows:

i. The input storage map is defined by the indices `lc_u` and `le_u` for u and `lc_v` and `le_v` for v:

```
lc_u = kr_loc(kr) + kt_loc(kt_u,kr)
le_u = lc_u + kt_len(kt_u,kr) - 1

lc_v = kr_loc(kr) + kt_loc(kt_v,kr)
le_v = lc_v + kt_len(kt_v,kr) - 1.
```

ii. The mapping indices `lc_u`, `le_u`, `lc_v`, and `lc_v` are also used to define indices to the IMAT tables **Aum_imat**, **Avm_imat**, **Aul_imat**, and **Avl_imat**:

```
lc_m = lc_u
le_m = le_u
lc_l = lc_v
le_l = le_v.
```

iii. The values of α_{um}, α_{vm}, α_{ul}, and α_{vl} are assigned from the IMAT tables **Aum_imat**, **Avm_imat**, **Aul_imat**, and **Avl_imat** respectively:

```
a_u(lc_m:le_m) =  (/ (Aum_imat(ktab(i),jtab(i)), i=lc_u,le_u) /)
a_v(lc_m:le_m) =  (/ (Avm_imat(ktab(i),jtab(i)), i=lc_v,le_v) /)
a_u(lc_l:le_l) =  (/ (Aul_imat(ktab(i),jtab(i)), i=lc_u,le_u) /)
a_v(lc_l:le_l) =  (/ (Avl_imat(ktab(i),jtab(i)), i=lc_v,le_v) /).
```

iv. Finally, the matrix-vector multiplication is applied to the input vector segment **Hvec** in the following loop:

```
do ivec=1,nvecs
  Xvec(lc_u:le_u, ivec) =                          &
    a_u(lc_m:le_m) * Hvec(lc_m:le_m, ivec) +       &
    a_u(lc_l:le_l) * Hvec(lc_l:le_l, ivec)

   Xvec(lc_v:le_v, ivec) =                         &
    a_v(lc_m:le_m) * Hvec(lc_m:le_m, ivec) +       &
    a_v(lc_l:le_l) * Hvec(lc_l:le_l, ivec)
end do.
```

The Transpose Γ^{h^T}: The transpose Γ^{h^T} is implemented in the routine `aj_Alf`:

```
subroutine aj_Alf( n_kr, kr_loc, kr_len,        &
                   n_kt, kt_loc, kt_len,        &
        nsize,  ktab,   jtab,                   &
        nvecs,   ldX,   Xvec,                   &
                 ldH,   Hvec                    ).
```

Table 27: Summary of arguments to `mv_Alf()`.

Variable	Type/Dimensions	Intent	Description
n_kr	INTEGER	IN	Number of Regions
kr_loc	INTEGER(n_kr)	IN	Index of Region Start
kr_len	INTEGER(n_kr)	IN	Region Lengths
n_kt	INTEGER	IN	Number of Regions
kt_loc	INTEGER(n_kt,n_kr)	IN	Start of each kt block for each region
kt_len	INTEGER(n_kt,n_kr)	IN	Length of each kt block for each region
nsize	INTEGER	IN	Dimension of Hvec and Xvec
ktab	INTEGER(nsize)	IN	Level reference for lookup tables
jtab	INTEGER(nsize)	IN	Latitude reference for lookup tables
nvecs	INTEGER	IN	Number of input/output vectors
ldH	INTEGER	IN	Dimension of each input vector **h**
Hvec	REAL(ldH,nvecs)	IN	Input vectors **h**
ldX	INTEGER	IN	Dimension of each output vector **x**
Xvec	REAL(ldX,nvecs)	OUT	Output vectors **x**

The arguments for `aj_Alf` are as in Table 27, with the exception that intents of `Xvec` and `Hvec` are now `IN` and `OUT`, respectively. The only substantial difference between this routine and `mv_Alf` is the matrix-vector multiplication. In `aj_Alf`, the matrix-vector multiplication is implemented using the following loop:

```
do ivec=1,nvecs
   Hvec(lc_m:le_m, ivec) =                          &
    a_u(lc_m:le_m) * Xvec(lc_u:le_u, ivec) +        &
    a_v(lc_m:le_m) * Xvec(lc_v:le_v, ivec)
   Hvec(lc_l:le_l, ivec) =                          &
    a_u(lc_l:le_l) * Xvec(lc_u:le_u, ivec) +        &
    a_v(lc_l:le_l) * Xvec(lc_v:le_v, ivec)
end do.
```

10.2 Mass-decoupled Height/Wind Error Covariances Γ^{ψ}, and Γ^{χ}

The mass-decoupled height/wind error covariance models used in the PSAS are described in [Guo et al., 1998] Sec. 4.

The global operator Γ^{ψ} is implemented in the routine `mv_Bet`:

```
subroutine mv_Bet( n_kr,kr_loc,kr_len,          &
                   n_kt,kt_loc,kt_len,          &
                   nsize,  ktab,   jtab,        &
                   nvecs,   ldH,   Hvec,        &
                   ldX,   Xvec                   ).
```

The argument list for **mv_Bet()** is identical to that for **mv_Alf()** – see Table 27. The indexing procedure using a temporary array **irow(1:nsize)** identical to that for **mv_Alf()**, resulting in an index table with **nrow** nonzero entries.

This operator is applied by looping over the segments of **Hvec** referenced by the array **irow(1:nrow)**, and acts only segments for which **kt** indicates a wind datatype.

For each wind segment, the operator

$$\hat{\beta}^{\psi} \equiv \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \tag{13}$$

is defined symbolically in [Guo et al., 1998] is applied to the input vector **Hvec**. Given region **kr**, this is done for each (u, v) by first calculating input storage map indices **lc_u** and **le_u** for u and **lc_v** and **le_v** for v:

```
lc_u = kr_loc(kr) + kt_loc(kt_u,kr)
le_u = lc_u + kt_len(kt_u,kr) - 1

lc_v = kr_loc(kr) + kt_loc(kt_v,kr)
le_v = lc_v + kt_len(kt_v,kr) - 1.
```

Output storage map indices to **Xvec** are then calculated:

```
lc_m = lc_u
le_m = le_u
lc_l = lc_v
le_l = le_v.
```

Finally, the matrix-vector multiplication is applied for each u and v segment in this region:

```
Xvec(lc_u:le_u, 1:nvecs) = -Hvec(lc_m:le_m, 1:nvecs)
Xvec(lc_v:le_v, 1:nvecs) =  Hvec(lc_l:le_l, 1:nvecs).
```

The implementation of the transpose operator Γ^{ψ^T} is the routine **aj_Bet()**:

```
subroutine aj_Bet( n_kr, kr_loc, kr_len,        &
                   n_kt, kt_loc, kt_len,        &
                   nsize, ktab,   jtab,         &
                   nvecs,  ldX,   Xvec,         &
                   ldH,   Hvec                   ).
```

The arguments to this routine are summarized in Table 27 with the single difference that the argument **Xvec** has intent **IN** and **Hvec** has intent **OUT**.

This routine is also similar to **mv_Bet**. The indexing scheme used above is employed, and the Γ^{ψ^T} acts only on **kt**-segments corresponding to wind variables. The matrix-vector multiply is implemented using:

```
Hvec(lc_m:le_m, 1:nvecs) = -Xvec(lc_u:le_u, 1:nvecs)
Hvec(lc_l:le_l, 1:nvecs) =  Xvec(lc_v:le_v, 1:nvecs).
```

The global operator Γ^x is implemented in the routine **mv_Gam**:

```
subroutine mv_Gam( n_kr,kr_loc,kr_len,          &
                   n_kt,kt_loc,kt_len,          &
                   nsize,  ktab,   jtab,        &
                   nvecs,   ldH,   Hvec,        &
                   ldX,   Xvec                   ).
```

The argument list for **mv_Gam()** is identical to that for **mv_Alf()** – see Table 27. The indexing procedure using a temporary array **irow(1:nsize)** identical to that for **mv_Gam()**, resulting in an index table with **nrow** nonzero entries.

The matrix-vector multiplication is applied by looping over the segments of **Hvec** referenced by the array **irow(1:nrow)**, and acts only segments for which **kt** indicates a wind datatype.

For each wind segment, the operator

$$\hat{\beta}^x \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{14}$$

is defined symbolically in [Guo et al., 1998] Sec. 4. is applied to the input vector **Hvec**. This is accomplished for each (u, v) pair in a given region **kr** by first calculating input storage map indices **lc_u** and **le_u** for u and **lc_v** and **le_v** for v:

```
lc_u = kr_loc(kr) + kt_loc(kt_u,kr)
le_u = lc_u + kt_len(kt_u,kr) - 1

lc_v = kr_loc(kr) + kt_loc(kt_v,kr)
le_v = lc_v + kt_len(kt_v,kr) - 1.
```

Output storage map indices to **Xvec** are then calculated:

```
lc_m = lc_u
le_m = le_u
lc_l = lc_v
le_l = le_v.
```

Finally, the matrix-vector multiplication is applied for each u and v segment in this region:

```
Xvec(lc_u:le_u, 1:nvecs) = Hvec(lc_l:le_l, 1:nvecs)
Xvec(lc_v:le_v, 1:nvecs) = Hvec(lc_m:le_m, 1:nvecs).
```

The transpose operator Γ^{xT} is implemented in the routine **aj_Gam()**:

```
subroutine aj_Gam( n_kr, kr_loc, kr_len,        &
                   n_kt, kt_loc, kt_len,        &
                   nsize,  ktab,   jtab,        &
                   nvecs,   ldX,   Xvec,        &
                   ldH,   Hvec                   )
```

The arguments to this routine are as in Table 27 with the single difference that the argument `Xvec` has intent `IN` and `Hvec` has intent `OUT`.

This routine is also similar to `mv_Bet`. The indexing scheme used above is employed, and the operator acts only on `kt`-segments corresponding to wind variables. The matrix-vector multiplication is implemented using:

```
Hvec(lc_m:le_m, 1:nvecs) = Xvec(lc_v:le_v, 1:nvecs)
Hvec(lc_l:le_l, 1:nvecs) = Xvec(lc_u:le_u, 1:nvecs).
```

10.3 Forecast/Observation Error Standard Deviation Operator $\bar{\Sigma}$

In this section, we describe the forecast error standard deviation operators Σ^h, Σ^ψ, Σ^x, and Σ^q, and the observation error standard deviation operators and Σ^o_u and Σ^o_c. These operators are applied to the vector `x` using the routine `mv_diag`:

```
subroutine mv_diag(n_kr,kr_loc,kr_len, n_kt,kt_loc,kt_len,      &
                   nsize, sigm, nvecs, ldx, x, ldy, y            ).
```

A summary of the arguments to `mv_diag()` is presented in Table 28. The diagonal operator $\bar{\Sigma}$ is represented by the data structure `sigm(1:nsize)`, which contains the `nsize` nonzero elements of $\bar{\Sigma}$. The input vectors `x`, represented by `x(1:ldx,1:nvecs)`, are scanned region-by-region, where regions are keyed using `kr_loc(kr)`. Within each region, these vectors are scanned type-by-type, where types are keyed using `kt_loc(kt,kr)`, to determine a `kr`/`kt` block of `x` whose initial index is `lc = kr_loc(kr) + kt_loc(kt,kr)` and final index is `le = lc + kt_len(kt,kr) - 1`. The operator $\bar{\Sigma}$ is applied to each vector segment in this block using:

```
do ivec=1,nvecs
  y(lc:le,ivec) = sigm(lc:le)*x(lc:le,ivec)
end do.
```

10.4 The Symmetric Error Correlation Operator = `sym_Cxpy()`

In order to solve the innovation equation (1), elements of forecast and observation error covariance matrices must be formed and subsequently act as operators on segments of the input vector `x` in Eqn. (1). The routine `sym_Cxpy()` constructs the matrices necessary to transform `x` into M`x`, where $M = \bar{H}P^f\bar{H}^T + R$ in the so-called *factored operator formulation* described in [Guo et al., 1998].

These matrices are formed, and the matrix-vector product M`x` is computed in the routine `sym_Cxpy()`. The flow control diagram of `sym_Cxpy()` is illustrated in Figure 51.

The interface to `sym_Cxpy()` is

```
subroutine sym_Cxpy(kind_mat, kind_cov, sparse,                  &
                    n_kr, kr_loc, kr_len, n_kt, kt_loc, kt_len,  &
                    n_x, kx, ks, ktab,                           &
                    qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y, &
                    nvecs, ldx, x, ldy, y, istat                  )
```

Table 28: Summary of arguments to `mv_diag()`.

Variable	Type/Dimensions	Intent	Description
`n_kr`	`INTEGER`	`IN`	Number of Regions
`kr_loc`	`INTEGER(n_kr)`	`IN`	Index of Region Start
`kr_len`	`INTEGER(n_kr)`	`IN`	Region Lengths
`n_kt`	`INTEGER`	`IN`	Number of Data Types
`kt_loc`	`INTEGER(n_kt,n_kr)`	`IN`	Start of each `kt` block for each region
`kt_len`	`INTEGER(n_kt,n_kr)`	`IN`	Length of each `kt` block for each region
`nsize`	`INTEGER`	`IN`	Dimension of Σ
`sigm`	`REAL(nsize)`	`IN`	Elements of Σ
`nvecs`	`INTEGER`	`IN`	Number of vectors $\mathbf{x}$
`ldx`	`INTEGER`	`IN`	Number of elements in each input vector $\mathbf{x}$
`x`	`REAL(ldx,nvecs)`	`IN`	Set of input vectors $\mathbf{x}$
`ldy`	`INTEGER`	`IN`	Number of elements in each output vector $\mathbf{y}$
`y`	`REAL(ldy,nvecs)`	`OUT`	Set of output vectors $\mathbf{y}$

and the description of its arguments is presented in Table 29.

The routine `sym_Cxpy()` performs the following steps:

1. Checks if input data satisfies the following conditions:
   (a) The leading dimension of the input vector `l_dx` $\leq$ `n_x`.
   (b) The leading dimension of the output vector `l_dy` $\leq$ `n_x`.
   (c) The matrix type flag `kind_mat` $\in$ {`kind_1mat`, `kind_2mat`, `kind_3mat`, `kind_4mat`, `kind_5mat`}.

   The error flag returns with value `istat` $= -1$ if any of these conditions are violated.

2. Allocation of workspace. For computational reasons, the matrices in Eqn. (1) are never explicitly formed in the software. Instead, the vector $\mathbf{x}$ is partitioned into segments (subvectors), and only those matrix elements needed to act on these segments are considered.

   The following are used as workspace arrays:

   (a) The initial indices of the `kr`/`kt` mesh elements `list_row(n_kt*n_kr)` and a status flag array `list_err(n_kt*n_kr)`. Both are `INTEGER` arrays.
   (b) `REAL` temporary arrays `Cxi(n_x,nvecs)` and `Cxj(n_x,nvecs)`. These arrays hold the matrix-vector product results; there are two of them to avoid memory contention on shared-memory parallel platforms.

Diagonal Band: We define a nonzero matrix block by the conditions:

- The number of rows of the block `kt_len`(`kt`, `kr`) > 0.
- The logical function `sparse(kind_mat,kind_cov,kr,kt,kr,kt)` is false.

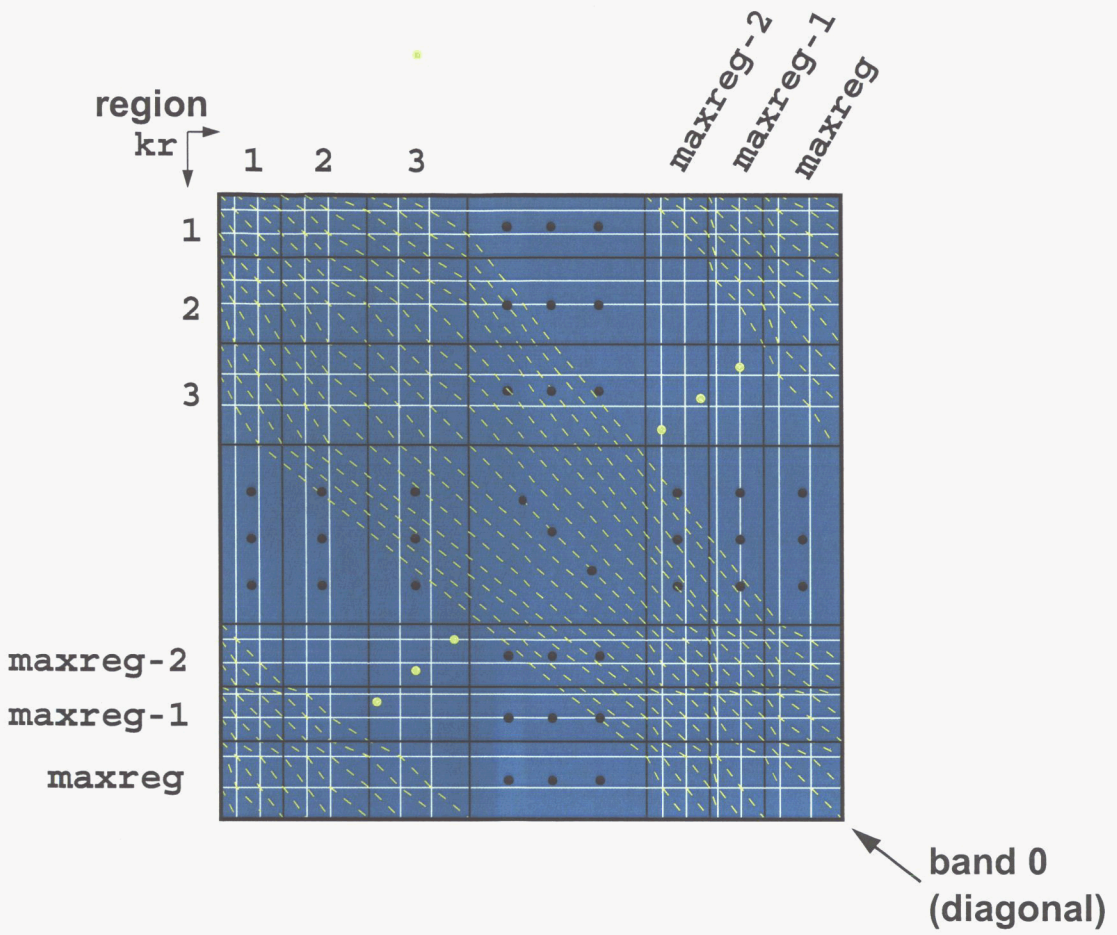


Figure 50: Band structure of matrix operators in PSAS.

The list of nonzero `kr/kt` blocks is constructed for the diagonal band by looping over the index `mrow = n_kr * n_kt`. For each nonzero block, the total number of blocks `nrow` is incremented, and the current row location `irow` is stored in `list_row(nrow)`.

The set of error flags in the array `list_err` is set to zero:

```
list_err(1:nrow) = 0 .
```

The application of the diagonal band of the symmetric operator is a loop over the `irow = 1, nrow` blocks indexed by `list_row`. For each value of `irow`, the local values of the region index `kr` and datatype index `kt` are calculated:

```
kr=  (irow-1)/n_kt + 1
kt = mod(irow-1,n_kt) + 1 .
```

Next, the starting index `lc` and ending index `le` are calculated for this block:

```
lc=kr_loc(kr) + kt_loc(kt,kr)
```

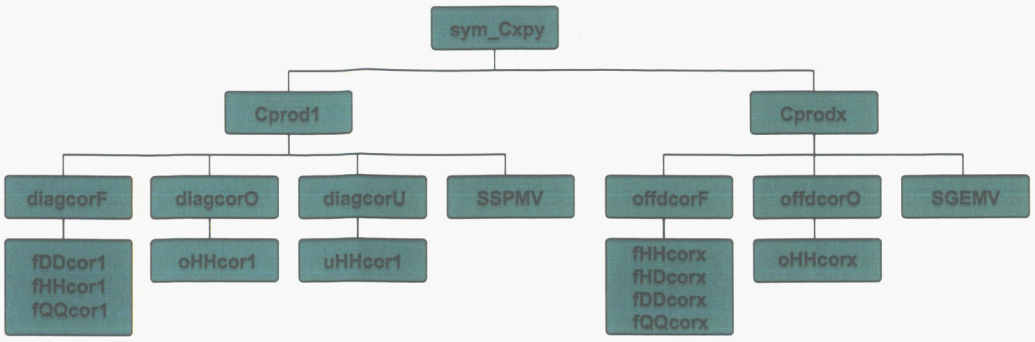


Figure 51: Control flow for `sym_Cxpy()`.

```
ln=kt_len(kt,kr)
le=lc+ln-1 .
```

The actual calculation of this block of the operator and its application to `x(lc:le,1:nvecs)` is a call to the routine `Cprod1()` described in Section 10.4.1 below:

```
call Cprod1(kind_cov,kr,kt,                          &
  ln, kx(lc),ks(lc),ktab(lc),              &
      qr_x(lc),qr_y(lc),qr_z(lc),          &
      qm_x(lc),qm_y(lc),qm_z(lc),          &
      ql_x(lc),ql_y(lc),                   &
  nvecs,  ldx, x(lc,1), n_x, Cxi(lc,1),    &
  ier                                      ).
```

The result of the matrix-vector multiplication is returned in the vector `Cxi(lc:le,1:nvecs)`, and the status flag `ier` for this block is stored in `list_err(irow)`. If `ier = 0`, then `Cprod1()` returned successfully, and the partial product `Cxi(lc:le,1:nvecs)` is added to the output vector `y`:

```
y(lc:le,1:nvecs) = Cxi(lc:le,1:nvecs) + y(lc:le,1:nvecs).
```

The type of operation requested is checked before proceeding to the routine described below that computes the off-diagonal bands of the operator. If the operator requested is the uncorrelated part of the observation error covariance matrix (R_u), then (`kind_mat` = `kind_Umat`), to indicate that there are no horizontal error correlations. In this case only the diagonal `kr/kt` band is computed, and `Cprod1()` returns.

Off-diagonal Bands: The action of the off-diagonal bands is a loop over the index `jband = 1,mband-1`, where `mband` is the number of bands in the operator.

There are two loops internal to this loop over j. The first internal loop over `irow = 1,mrow` indexes the so-called nonzero blocks. The number of nonzero block is set to `nrow` in this loop. The second internal loop over `ir = 1,nrow` calls the routine `Cprodx` to carry out the covariance matrix-vector multiplication for each of the nonzero blocks located in the first internal loop.

If the operator is the correlated part of the observation error covariance matrix (R_c), then `kind_mat` = `kind_Rmat`, to indicate that the error correlation model is currently univariate

and the number of nonzero bands is `mband` = `n_kt`. For `kind_mat` $\neq$ `kind_Rmat`, the number of nonzero bands is set to `mband` = `n_kr` $*$ `n_kt`.

For each band, the maximum number of row segments `mrow` is set, and the nonzero operator block counter `nrow` is initialized:

```
mrow = n_kt*n_kr - jband
nrow = 0.
```

The action of off-diagonal bands is implemented in Steps **I** and **II** below.

I. Indexing of nonzero bands:

The list of nonzero `kr`/`kt` blocks is constructed for this band is accomplished by looping over the index `irow = 1,mrow`. The local column index `jcol` is defined and the dimensions of each block are calculated:

```
jcol = irow + jband.
```

Row indices/dimensions:

```
kr_i = (irow-1)/n_kt + 1
kt_i = mod(irow-1,n_kt) + 1
ln_i = kt_len(kt_i,kr_i).
```

Column indices/dimensions:

```
kr_j = (jcol-1)/n_kt + 1
kt_j = mod(jcol-1,n_kt) + 1
ln_j = kt_len(kt_j,kr_j).
```

The criteria for a nonzero block are:

- The number of rows in the block `ln_i` > 0.
- The number of columns in the block `ln_j` > 0.
- The logical function `sparse(kind_mat,kind_cov,kr_i,kt_i,kr_j,kt_j)` is false.

For each nonzero block, the total number of blocks `nrow` is incremented, and the current row location `irow` is stored in `list_row(nrow)`.

At the end of the sweep through this band, the value of `nrow` is checked. If `nrow = 0`, there are no nonzero operator blocks in this band, and the calculation jumps to the next band. If there exist nonzero blocks in this band, then the status vector `list_err` and partial product arrays `Cxi` and `Cxj` are initialized:

```
list_err(1:nrow) = 0
Cxi(1:nrow) = 0.0
Cxj(1:nrow) = 0.0  .
```

II. Covariance matrix-vector multiplication of nonzero blocks - Cprodx:

The application of the off-diagonal blocks of the operator for this band are a loop over the index `ir = 1,nrow`. For each nonzero block in the band, indices of its location are calculated:

```
irow = list_row(ir)
jcol = irow + jband.
```

For each nonzero block, the region and datatype row indices `kr_i` and `kt_i` are calculated, along with row pointer `lc_i` and block row dimension `ln_i`:

```
kr_i = (irow-1)/n_kt + 1
kt_i = mod(irow-1,n_kt) + 1
lc_i = kr_loc(kr_i) + kt_loc(kt_i,kr_i)
ln_i = kt_len(kt_i,kr_i).
```

Region and datatype column indices `kr_j` and `kt_j` are calculated, along with column pointer `lc_j` and block column dimension `ln_j`:

```
kr_j = (jcol-1)/n_kt + 1
kt_j = mod(jcol-1,n_kt) + 1
lc_j = kr_loc(kr_j) + kt_loc(kt_j,kr_j)
ln_j = kt_len(kt_j,kr_j).
```

Once the location and dimensions of this operator block are determined, the operator is calculated and applied using the routine `Cprodx()` described in Section 10.4.2 below:

```
call Cprodx(kind_cov,                                  &
  kr_i,kt_i,                                           &
  ln_i,   kx(lc_i),                                    &
        ktab(lc_i),                              &
        qr_x(lc_i),qr_y(lc_i),qr_z(lc_i),              &
        qm_x(lc_i),qm_y(lc_i),qm_z(lc_i),              &
        ql_x(lc_i),ql_y(lc_i),                         &
  kr_j,kt_j,                                           &
  ln_j,   kx(lc_j),                                    &
        ktab(lc_j),                              &
        qr_x(lc_j),qr_y(lc_j),qr_z(lc_j),              &
        qm_x(lc_j),qm_y(lc_j),qm_z(lc_j),              &
        ql_x(lc_j),ql_y(lc_j),                         &
  nvecs,  ldx, x(lc_i,1), n_x, Cxi(lc_j,1),            &
          ldx, x(lc_j,1), n_x, Cxj(lc_i,1),            &
  ier                                                  ).
```

The value of the error flag `ier` is stored in `list_err(ir)`. Once the loop over the blocks in this band is complete, the status array `list_err(1:nrow)` is scanned for nonzero elements. If any are found, `sym_Cxpy()` returns with a nonzero value of its status flag `istat`.

If the application of the operator blocks were all successful for this band, then the output vector `y` is updated with the results stored in `Cxi` and `Cxj`:

```
y(1:n_x,1:nvecs) = y(1:n_x,1:nvecs) + Cxi(1:n_x,1:nvecs)
                                    + Cxj(1:n_x,1:nvecs).
```

Once the calculation is completed for all the bands requested, the local arrays `list_row`, `list_err`, `Cxi`, and `Cxj` are deallocated, and `sym_Cxpy()` returns.

10.4.1 The Diagonal Block Operator `Cprod1()`

The calculation and application of the individual blocks in the diagonal band of symmetric operators is implemented in the routine `Cprod1()`:

```
subroutine Cprod1(kind_cov,                                    &
                  kr,kt, ln,   kx,  ks,ktab,                   &
                  qr_x,qr_y,qr_z,qm_x,qm_y,qm_z,ql_x,ql_y, &
                  nvecs,  ldX, x, ldCx, Cx, ierr                ).
```

The arguments to `Cprod1()` are described in Table 31.

The following conditions are tested in the initial step in `Cprod1()`:

- The input vector leading dimension `ldX` $\geq$ `ln`.
- The output vector leading dimension `ldCx` $\geq$ `ln`.
- The requested covariance type `kind_cov` is one of the following: `kind_covF`, `kind_covS`, `kind_covV`, `kind_cov0`, `kind_covU`.

If any of above conditions is violated, `Cprod1()` returns with `ierr` $= -1$. If `nvecs` $= 0$ or `ln` $= 0$, no action is necessary and `Cprod1()` returns with `ierr` $= 0$.

The operator blocks for the diagonal band are symmetric, and thus the `ln * ln` block is represented in upper triangular form in a dynamically allocated array `Corr(ln*(ln+1)/2)`. The elements of the operator are calculated by a call to the appropriate error correlation module. This decision is made based on the value of the variable `kind_cov`. The routines called are shown in Table 30.

Each of these functions returns:

- The type of correlation block in the `CHARACTER` variable `Mtyp`: `'U'` or `'u'` for upper triangular, `'I'` or `'i'` for the identity, or `'E'` if an error occurred.
- The packed operator block `Corr(ln*(ln+1)/2)`.

If `Mtyp` has value `'I'` or `'i'`, then each output vector in `Cx` is assigned the respective input values from `x`:

```
Cx(1:ln,1:nvecs)=x(1:ln,1:nvecs).
```

If `Mtyp` has value `'U'` or `'u'`, then the each of the vectors in the output structure `Cx` is calculated using a call to the BLAS function `SSPMV`:

```
do ivec=1,nvecs
   call SSPMV('U',ln,1.,Corr,x(1,ivec),1,0.,Cx(1,ivec),1)
end do.
```

The temporary structure `Corr` is deallocated and `Cprod1()` returns.

Table 29: Summary of arguments to `sym_Cxpy()`.

Variable	Type/Dimensions	Intent	Description
kind_mat	INTEGER	IN	Matrix Type
kind_cov	INTEGER	IN	Covariance Type
sparse	INTEGER	IN	Matrix Type
n_kr	INTEGER	IN	Number of Regions
kr_loc	INTEGER(n_kr)	IN	Index of Region Start
kr_len	INTEGER(n_kr)	IN	Region Lengths
n_kt	INTEGER	IN	Number of Data Types
kt_loc	INTEGER(n_kt,n_kr)	IN	Start of each kt block for each region
kt_len	INTEGER(n_kt,n_kr)	IN	Length of each kt block for each region
n_x	INTEGER	IN	Dimension of Attribute Arrays
kx	INTEGER(n_x)	IN	Data Source Index
ks	INTEGER(n_x)	IN	Sounding Index
ktab	INTEGER(n_x)	IN	Level Index for Look-Up Tables
qr_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$
qr_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$
qr_z	REAL(n_x)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$
qm_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$
qm_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$
qm_z	REAL(n_x)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$
ql_x	REAL(n_x)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$
ql_y	REAL(n_x)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$
nvecs	INTEGER	IN	Number of vectors
ldx	INTEGER	IN	Leading dimension of x
x	REAL(ldx,nvecs)	IN	Input vectors x
ldy	INTEGER	IN	Leading dimension of y
y	REAL(ldy,nvecs)	OUT	Output vectors y
istat	INTEGER	OUT	Return Status

Table 30: Error Correlation Calculation Calls from `Cprod1()`.

Value of kind_cov	Routine Called	Error Types
kind_covF	diagcorF()	Forecast h, Mass-coupled (u, v), q
kind_covS	diagcorF()	Forecast Mass-decoupled Wind (ψ)
kind_covV	diagcorF()	Forecast Mass-decoupled Wind (χ)
kind_covC	diagcor0()	Horizontally Correlated Obs.
kind_covU	diagcorU()	Horizontally Uncorrelated Obs.

Table 31: Summary of arguments to `Cprod1()`.

Variable	Type/Dimensions	Intent	Description
kind_cov	INTEGER	IN	Covariance Type
kr	INTEGER	IN	Region Index
kt	INTEGER	IN	Datatype Index
ln	INTEGER	IN	Block Dimension
kx	INTEGER(ln)	IN	Data Source Index (observations)
ks	INTEGER(ln)	IN	Sounding Index (observations)
ktab	INTEGER(ln)	IN	Look-up Table Level Indices
qr_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$
qr_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$
qr_z	REAL(ln)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$
qm_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$
qm_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$
qm_z	REAL(ln)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$
ql_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$
ql_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$
nvecs	INTEGER	IN	Number of vectors
ldX	INTEGER	IN	Leading dimension of **x**
x	REAL(ldX,nvecs)	IN	Input vectors **x**
ldCx	INTEGER	IN	Leading dimension of C **x**
Cx	REAL(ldCx,nvecs)	OUT	Output vectors $C\mathbf{x}$
ierr	INTEGER	OUT	Return Status

10.4.2 The Off-diagonal Block Operator `Cprodx()`

The calculation and application of the individual blocks in the off-diagonal bands of symmetric operators is implemented in the routine `Cprodx()`:

```
subroutine Cprodx(kind_cov,                &
                  kri,kti,leni,kxi,ktabi,  &
                  qri_x,qri_y,qri_z,       &
                  qmi_x,qmi_y,qmi_z,       &
                  qli_x,qli_y,             &
                  krj,ktj,lenj,kxj,ktabj,  &
                  qrj_x,qrj_y,qrj_z,       &
                  qmj_x,qmj_y,qmj_z,       &
                  qlj_x,qlj_y,             &
                  nvecs,ldXi,xi,ldCxi,Cxi, &
                  ldXj,xj,ldCxj,Cxj,       &
                  ierr                      ).
```

The arguments to `Cprodx()` are described in Table 33.

The following conditions on input data are tested in the initial step in `Cprodx()`:

- Input vector i-segment length `ldXi` $\geq$ `leni`.
- Output vector j-segment length `ldCxj` $\geq$ `leni`.
- Input vector j-segment length `ldXj` $\geq$ `lenj`.
- Output vector i-segment length `ldCxi` $\geq$ `lenj`.
- The type of error correlation block requested corresponding to the variable `kind_cov` is one of the following: `kind_covF`, `kind_covS`, `kind_covV`, or `kind_covC`.

If any of the above conditions is violated, `Cprodx()` returns with error flag `ierr` $= -1$.

There are some special execution cases that require at most direct assignment of output data:

- If `leni` and `lenj` are both zero, no action is necessary; `Cprodx()` returns.
- If `leni` $= 0$ and `lenj` > 0, set the output vector segments `Cxi(1:lenj,1:nvecs)` and return.
- If `lenj` $= 0$ and `leni` > 0, set the output vector segments `Cxj(1:leni,1:nvecs)` and return.

The error correlation block to be computed is stored in the dynamically allocated array `Corr(leni*lenj)`. The elements of the operator block are calculated by a call to the appropriate error correlation module. The module called is determined by the variable `kind_cov`. The routines called and their types are summarized in Table 32.

The error correlation operator values are returned in `Corr`, along with the matrix type stored in a `CHARACTER` variable `Mtyp`. There are three possible classes of values for `Mtyp`: `'T'` or `'t'` for a transpose block, `'N'` or `'n'` for a non-transpose block, and `'Z'` for a zero block.

Table 32: Error Correlation Calculation Calls from `Cprodx()`.

Value of `kind_cov`	Routine Called	Error Types
`kind_covF`	`offdcorF()`	Forecast h, Mass-coupled (u, v), q
`kind_covS`	`offdcorF()`	Forecast Mass-decoupled Wind (ψ)
`kind_covV`	`offdcorF()`	Forecast Mass-decoupled Wind (χ)
`kind_covC`	`offdcor0()`	Horizontally Correlated Obs.

The application of the block operator and its transpose to each of the `nvec` input vectors proceeds in a loop over the index `ivec`, with the matrix-vector multiplication performed by a call to the BLAS function `SGEMV()`.

Suppose that the operator block stored in `Corr` is a transposed block. The following pair of calls to `SGEMV()` perform matrix-vector segment multiplications of an upper triangular block within `Corr` and of its transpose residing in the lower triangular block of `Corr`:

```
call SGEMV('T',lenj,leni,1.,Corr,lenj, xj(1,ivec),1, &
                                0.,          Cxj(1,ivec),1)
call SGEMV('N',lenj,leni,1.,Corr,lenj, xi(1,ivec),1, &
                                0.,          Cxi(1,ivec),1).
```

Suppose that the operator block stored in `Corr` is a non-transposed block. The following pair of calls to `SGEMV()` perform matrix-vector segment multiplications of an upper triangular block within `Corr` and of its transpose residing in the lower triangular block of `Corr`:

```
call SGEMV('N',leni,lenj,1.,Corr,leni, xj(1,ivec),1, &
                                0.,          Cxj(1,ivec),1)
call SGEMV('T',leni,lenj,1.,Corr,leni, xi(1,ivec),1, &
                                0.,          Cxi(1,ivec),1).
```

For zero blocks, no multiplication is necessary:

```
Cxi(1:lenj,ivec)=0.
Cxj(1:leni,ivec)=0.
```

Upon completion of the multiplications for each vector, `Corr` is deallocated and `Cprodx()` returns with `ierr` $= 0$.

10.5 The Rectangular Error Correlation Operator C_{rec}

The rectangular error correlation operator C_{rec} is used to calculate the forecast error covariance operator P^f in the mixed forecast/observation representation. The implementation of this class of operator is the routine `rec_Cxpy`:

Table 33: Summary of arguments to `Cprodx()`.

Variable	Type/Dimensions	Intent	Description
kind_cov	INTEGER	IN	Covariance Type
kri	INTEGER	IN	Region Index (rows)
kti	INTEGER	IN	Datatype Index (rows)
leni	INTEGER	IN	Block Row Dimension
kxi	INTEGER	IN	Data Source Index (rows)
ktabi	INTEGER(ln)	IN	Look-up Table Level Indices (rows)
qri_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qri_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qri_z	REAL(ln)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qmi_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qmi_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qmi_z	REAL(ln)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qli_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$ (rows)
qli_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$ (rows)
krj	INTEGER	IN	Region Index (columns)
ktj	INTEGER	IN	Datatype Index (columns)
lenj	INTEGER	IN	Block Row Dimension
kxj	INTEGER	IN	Data Source Index (columns)
ktabj	INTEGER(ln)	IN	Look-up Table Level Indices (columns)
qrj_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$ (columns)
qrj_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$ (columns)
qrj_z	REAL(ln)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$ (columns)
qmj_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$ (columns)
qmj_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$ (columns)
qmj_z	REAL(ln)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$ (columns)
qlj_x	REAL(ln)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$ (columns)
qlj_y	REAL(ln)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$ (columns)
nvecs	INTEGER	IN	Number of vectors
ldXi	INTEGER	IN	Leading dimension of **x**
xi	REAL(ldX,nvecs)	IN	Input vectors for block ji
ldCxi	INTEGER	IN	Leading dimension of Cxi
Cxi	REAL(ldCx,nvecs)	OUT	Output vectors $C_{ji}\mathbf{x}$
ldXj	INTEGER	IN	Leading dimension of **x**
xj	REAL(ldX,nvecs)	IN	Input vectors for block ij
ldCxj	INTEGER	IN	Leading dimension of C **x**
Cxj	REAL(ldCx,nvecs)	OUT	Output vector segment $C_{ij}\mathbf{x}$
ierr	INTEGER	OUT	Return Status

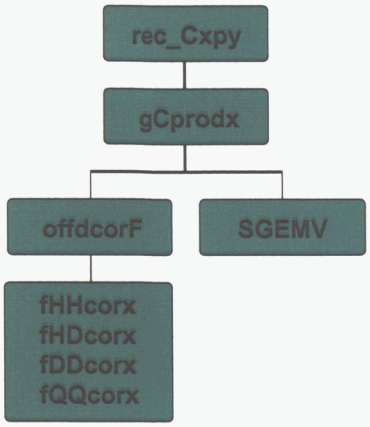


Figure 52: Calling tree for the operator `rec_Cxpy()`.

```
subroutine rec_Cxpy(kind_mat, kind_cov, sparse,              &
     n_kri, kr_loci,kr_leni, n_kti, kt_loci,kt_leni,         &
     n_i,   ktabi,                                           &
          qri_x,qri_y,qri_z,qmi_x,qmi_y,qmi_z,qli_x,qli_y,   &
     n_krj, kr_locj,kr_lenj, n_ktj, kt_locj,kt_lenj,         &
     n_j,   ktabj,                                           &
          qrj_x,qrj_y,qrj_z,qmj_x,qmj_y,qmj_z,qlj_x,qlj_y,   &
     nvecs, ldxj, xj, ldyi, yi, istat                        ).
```

The arguments to `rec_Cxpy` are summarized in Table 34, and its procedural calling tree is illustrated in Figure 52.

The following checks are first applied to input data:

- Output vector attribute lengths and vector lengths are compatible: `n_i` $\leq$ `ldyi`.
- Input vector attribute lengths and vector lengths are compatible: `n_j` $\leq$ `ldxj`.
- The requested operator type `kind_mat` $\in$ {`kind_Umat`, `kind_Rmat`, `kind_3mat`, `kind_4mat`, `kind_5mat`}.

If any of the above conditions are violated, `rec_Cxpy()` returns with `istat` $= -1$.

Temporary workspace is allocated:

- `list_row(n_ktj*n_krj)`, an `INTEGER` list of operator blocks.
- `list_err(n_ktj*n_krj)`, an `INTEGER` success status for calculation/application of individual operator blocks.
- `correlated(n_kti*n_kri, n_ktj*n_krj)`, a `LOGICAL` table of flags determining whether or not a particular block needs to be calculated.
- `Cxj(n_i,nvecs)`, a set of temporary result vectors.

Once workspace is prepared, the set of blocks that need to be computed for this operator is determined. The number of nonzero blocks `nrow` is initialized to zero, and the block list

is built using a nested scan of the rows i and within each row the columns j. The rows are scanned is a loop over `irow=1,n_kti*n_kri`. The local region and datatype index for `irow` are calculated, as well as the length of this particular `kr/kt` segment of the rows:

```
kri=   (irow-1)/n_kti +1
kti=mod(irow-1, n_kti)+1
lni=kt_leni(kti,kri).
```

If the row segment dimension `lni` is nonpositive, the calculation proceeds to the next value of `irow`. Within each row segment, the columns are scanned over the index `jcol=1,n_ktj*n_krj`. The local region and datatype index corresponding to `jcol` are calculated, as well as the length of this particular `kr/kt` segment of the columns:

```
krj=   (jcol-1)/n_ktj +1
ktj=mod(jcol-1, n_ktj)+1
lnj=kt_lenj(ktj,krj).
```

If the column segment dimension `lnj` is nonpositive, the calculation proceeds to the next value of `jcol`. If `lnj` > 0, then this block is evaluated based on the value of the array:

```
correlated(irow,jcol) = .not. sparse(kind_mat,kind_cov, kri,kti, krj,ktj).
```

Once the scan over `jcol` is complete, a second scan over `jcol` is performed to build the list of nonzero blocks:

```
do jcol=1,n_ktj*n_krj
  if(correlated(irow,jcol)) then
    nrow=nrow+1
    list_row(nrow)=irow
    exit
  endif
end do.
```

This completes the row and column scan. The result is a set of row segment pointer indices `list_row(1:nrow)`. A final preparation step is to initialize the error flag array `list_err(1:nrow) = 0`.

Calculation/Application of the Operator: The calculation and subsequent application of the operator proceeds as a loop over the index `ir` $= 1,$ `nrow` nonzero blocks. For each value of `ir` the following parameters are evaluated:

- the row segment `irow = list_row(ir)`.
- the row region index `kri = (irow-1)/n_kti + 1`.
- the row datatype index `kti = mod(irow-1, n_kti) + 1`.
- the starting index in row of the block `lci = kr_loci(kri) + kt_loci(kti,kri)`.
- the extent in rows of the block `lni = kt_leni(kti,kri)`.
- the ending index in rows of the block `lei = lci + lni - 1`.

Within this strip in rows, there is a scan for nonzero column blocks over the index `jcol = 1,n_ktj*n_krj`. The following column parameters for each block are computed:

- the column region index `krj = (jcol-1)/n_ktj + 1`.
- the column datatype index `ktj=mod(jcol-1, n_ktj)+1`.
- the starting index in columns for the block `lcj = kr_locj(krj) + kt_locj(ktj,krj)`.
- the extent in columns of the block `lnj = kt_lenj(ktj,krj)`.

A check is made to assure `correlated(irow,jcol) = .true.`; that is, this block needs to be calculated and applied. If this check is passed, the calculation of the block and its subsequent application are performed using a call to `gCprodx()`:

```
call gCprodx(kind_cov,                                    &
             kri,kti,                                     &
             lni,ktabi(lci),                              &
             qri_x(lci),qri_y(lci),qri_z(lci),            &
             qmi_x(lci),qmi_y(lci),qmi_z(lci),            &
             qli_x(lci),qli_y(lci),                       &
             krj,ktj,                                     &
             lnj,ktabj(lcj),                              &
             qrj_x(lcj),qrj_y(lcj),qrj_z(lcj),            &
             qmj_x(lcj),qmj_y(lcj),qmj_z(lcj),            &
             qlj_x(lcj),qlj_y(lcj),                       &
             nvecs,  ldxj, xj(lcj,1), n_i, Cxj(lci,1), &
             ier                                           ).
```

The result of the application of this block of the operator to `x` is stored in `Cxj(lci:lei,1:nvecs)`. The error flag `ier` is stored in `list_err(ir)`, and if it is nonzero, the routine exits the loop. If `ier` $= 0$, the operation was successful, and the output vector `yi` is updated accordingly:

```
yi(lci:lei,1:nvecs)=yi(lci:lei,1:nvecs)+Cxj(lci:lei,1:nvecs).
```

Outside of the evaluation loop, there is a check for nonzero values of `list_row`. If any exist, a message is written to `stderr`.

After the calculation is complete, the temporary arrays `list_err`, `list_row`, and `Cxj` are deallocated, and `rec_Cxpy()` returns.

10.5.1 The Block Operator `gCprodx()`

The actual calculation and application of the block operators discussed in Section 10.5 is accomplished by the routine `gCprodx()`:

```
subroutine gCprodx( kind_cov,                             &
                    kri,kti,  leni, ktabi,                &
                    qri_x,qri_y,qri_z,                    &
                    qmi_x,qmi_y,qmi_z,                    &
                    qli_x,qli_y,                          &
                    krj,ktj,lenj,ktabj,                   &
```

Table 34: Summary of arguments to `rec_Cxpy()`. The 'columns' arguments `n_krj`, ..., `qlj_y` are not shown for brevity.

Variable	Type/Dimensions	Intent	Description
kind_mat	INTEGER	IN	Matrix Type
kind_cov	INTEGER	IN	Covariance Type
sparse	INTEGER	IN	Matrix Type
n_kri	INTEGER	IN	Number of Regions (rows)
kr_loci	INTEGER(n_kr)	IN	Region Start (rows)
kr_leni	INTEGER(n_kr)	IN	Region Lengths (rows)
n_kti	INTEGER	IN	Number of Data Types (rows)
kt_loci	INTEGER(n_kt,n_kr)	IN	Start of each **kt** block (rows)
kt_leni	INTEGER(n_kt,n_kr)	IN	Length of each **kt** block (rows)
n_i	INTEGER	IN	Attribute Array Dimension (rows)
ktabi	INTEGER(n_i)	IN	Level Index for Look-Up Tables (rows)
qri_x	REAL(n_i)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qri_y	REAL(n_i)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qri_z	REAL(n_i)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qmi_x	REAL(n_i)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qmi_y	REAL(n_i)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qmi_z	REAL(n_i)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qli_x	REAL(n_i)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$ (rows)
qli_y	REAL(n_i)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$ (rows)
nvecs	INTEGER	IN	Number of vectors
ldxj	INTEGER	IN	Leading dimension of **x**
xj	REAL(ldxj,nvecs)	IN	Input vectors x
ldyi	INTEGER	IN	Leading dimension of **y**
yi	REAL(ldyi,nvecs)	OUT	Output vectors y
istat	INTEGER	OUT	Return Status

```
                    qrj_x,qrj_y,qrj_z,              &
                    qmj_x,qmj_y,qmj_z,              &
                    qlj_x,qlj_y,                    &
                    nvecs,ldXj,xj,ldCxj,Cxj,        &
                    ierr                             ).
```

The arguments of **gCprodx()** are summarized in Table 35.

The first step taken in **gCprodx()** is a set of sanity checks on its arguments:

- **ldXj** $\geq$ **lenj** and **ldCxj** $\geq$ **leni**.
- **kind_cov** $\in$ {**kind_covF**, **kind_covS**, **kind_covV**}

If any of these conditions are violated, **gCprodx()** returns with **ierr** $= -1$. If either the number of vectors **nvecs** or the block row dimension **leni** are nonpositive, **gCprodx()** returns. In the case of a block with **lenj** $= 0$, the resultant operator product **Cxj** for this block is zero:

```
Cxj(1:leni,1:nvecs) = 0.
```

The block of the error correlation operator is stored in dynamic memory in the array **Corr(1:leni*lenj)**. This block is calculated through a call to **offdcorF()**:

```
call offdcorF(kind_cov,                         &
              kti,leni,                         &
              qri_x,qri_y,qri_z,                &
              qmi_x,qmi_y,qmi_z,                &
              qli_x,qli_y,                      &
              ktabi,                            &
              ktj,lenj,                         &
              qrj_x,qrj_y,qrj_z,                &
              qmj_x,qmj_y,qmj_z,                &
              qlj_x,qlj_y,                      &
              ktabj,                            &
              Mtyp,    Corr, ierr               ).
```

The operator block is returned in the structure **Corr(1:leni*lenj)**. The type of block calculated is returned in the **CHARACTER** variable **Mtyp**, which has value **'T'** for a transposed block, **'N'** for a non-transposed block, and **'E'** in the event of an error.

If **Mtyp** $\in$ {'T', 't', 'N', 'n'}, the application of the forecast error correlation operator block to each of the input vectors in **xj** is done via a set of calls to the BLAS routine **SGEMV**:

```
do ivec=1,nvecs
   call SGEMV(Mtyp,lenj,leni,1.,Corr,lenj, xj(1,ivec),1, &
                              0.,          Cxj(1,ivec),1)
end do.
```

The result of the product returned in **Cxj(1:ldCxj,1:nvecs)**.

Table 35: Summary of arguments to `gCprodx()`.

Variable	Type/Dimensions	Intent	Description
kind_cov	INTEGER	IN	Covariance Type
kri	INTEGER	IN	Region index (rows)
kti	INTEGER	IN	Datatype index (rows)
leni	INTEGER	IN	Row Length
ktabi	INTEGER(leni)	IN	Level Index (rows)
qri_x	REAL(leni)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qri_y	REAL(leni)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qri_z	REAL(leni)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qmi_x	REAL(leni)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qmi_y	REAL(leni)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qmi_z	REAL(leni)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qli_x	REAL(leni)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$ (rows)
qli_y	REAL(leni)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$ (rows)
krj	INTEGER	IN	Region index (rows)
ktj	INTEGER)	IN	Datatype index (rows)
lenj	INTEGER	IN	Row Length
ktabj	INTEGER(lenj)	IN	Level Index (rows)
qrj_x	REAL(lenj)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qrj_y	REAL(lenj)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qrj_z	REAL(lenj)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{r}$ (rows)
qmj_x	REAL(lenj)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qmj_y	REAL(lenj)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qmj_z	REAL(lenj)	IN	z-component of $\hat{\mathbf{e}}_\mathbf{m}$ (rows)
qlj_x	REAL(lenj)	IN	x-component of $\hat{\mathbf{e}}_\mathbf{l}$ (rows)
qlj_y	REAL(lenj)	IN	y-component of $\hat{\mathbf{e}}_\mathbf{l}$ (rows)
nvecs	INTEGER	IN	Number of vectors
ldXj	INTEGER	IN	Leading dimension of **x**
xj	REAL(ldxj,nvecs)	IN	Input vectors x
ldCxj	INTEGER	IN	Leading dimension of **y**
Cxj	REAL(ldyi,nvecs)	OUT	Output vectors y
ierr	INTEGER	OUT	Return Status

Appendix A: The Analysis Interfaces `getAIall()`

The Sea-level Pressure/Wind Analysis `getAIpuv()` The PSAS interface that computes only sea-level pressure/wind analysis increments is the routine **`getAIpuv()`**, whose interface is shown below.

```
subroutine getAIpuv( npieces, lat_list, lon_list, pres_list, &
                     time_list, kx_list, kt_list, dels_list, &
                     sigF_list, sigO_list,                   &
                     im, jnp,                                &
                     psl_sigF, usl_sigF, vsl_sigF,           &
                     psl_inc, usl_inc, vsl_inc,              &
                     psl_sigA, usl_sigA, vsl_sigA             )
```

The calling tree for **`getAIpuv()`** is shown in Figure 9, and its arguments are summarized in Table 36.

This routine is modified from **`getAIall()`** in the following way:

- In order to keep the same interface to some routines called in the analysis (e.g., **`getAinc()`**), some of the arrays are *vestigial*; that is, they are dimensioned as scalars. These arrays are: **`z_sigF(1,1,1)`**, **`u_sigF(1,1,1)`**, **`v_sigF(1,1,1)`**, **`mix_sigF(1,1,1)`**, **`z_inc(1,1,1)`**, **`u_inc(1,1,1)`**, **`v_inc(1,1,1)`**, **`mix_inc(1,1,1)`**, **`z_sigA(1,1,1)`**, **`u_sigA(1,1,1)`**, **`v_sigA(1,1,1)`**, and **`mix_sigA(1,1,1)`**.
- The **`LOGICAL`** parameters **`want_z`**, **`want_u`**, **`want_v`**, and **`want_mix`** are set to **`.FALSE.`**
- Since no upper-air AI's are to be calculated, calls to the routines **`dervsigF_upD()`** and **`dervsigF_upW()`** are not made.

Table 36: Data passed into `getAIpuv()` via its interface.

Variable	Type	Intent	Description
npieces	INTEGER	INOUT	Number of Innovations
lat_list	REAL(npieces)	INOUT	Latitude
lon_list	REAL(npieces)	INOUT	Longitude
pres_list	REAL(npieces)	INOUT	Pressure (hPa)
time_list	REAL(npieces)	INOUT	Δt (min) from Analysis Time
kx_list	INTEGER(npieces)	INOUT	Data Source Index **kx**
kt_list	INTEGER(npieces)	INOUT	Data Type Index **kt**
dels_list	REAL(npieces)	INOUT	Innovation O-F
sigF_list	REAL(npieces)	INOUT	Forecast Error Variances σ_f
sigO_list	REAL(npieces)	INOUT	Observation Error Variances σ_o
im	INTEGER	IN	Number of Longitudinal Grid Points
jnp	INTEGER	IN	Number of Latitudinal Grid Points
psl_sigF	REAL(im,jnp)	IN	Forecast Sea-Level Pressure Error Variances
usl_sigF	REAL(im,jnp)	IN	Forecast u_{sl} Error Variances
vsl_sigF	REAL(im,jnp)	IN	Forecast v_{sl} Error Variances
psl_inc	REAL(im,jnp)	OUT	Sea-Level Pressure Analysis Increments
usl_inc	REAL(im,jnp)	OUT	u_{sl} Analysis Increments
vsl_inc	REAL(im,jnp)	OUT	v_{sl} Analysis Increments
psl_sigA	REAL(im,jnp)	OUT	Sea-Level Pressure Analysis Error Variances
usl_sigA	REAL(im,jnp)	OUT	u_{sl} Analysis Error Variances
vsl_sigA	REAL(im,jnp)	OUT	v_{sl} Analysis Error Variances

The Upper-air Height/Wind Analysis `getAIzuv()` The PSAS interface that computes only upper-air height/wind analysis increments is the routine **`getAIzuv()`**, whose interface is shown below.

```
subroutine getAIzuv( npieces, lat_list, lon_list, pres_list, &
                     time_list, kx_list, kt_list, dels_list, &
                     sigF_list, sig0_list,                   &
                     im, jnp, mlev, pres_lev,                &
                     z_sigF, u_sigF, v_sigF,                 &
                     z_inc, u_inc, v_inc,                    &
                     z_sigA, u_sigA, v_sigA                   )
```

The calling tree for **`getAIzuv()`** is shown in Figure 9, and its arguments are summarized in Table 37.

This routine is modified from **`getAIall()`** in the following way:

- In order to keep the same interface to some routines called in the analysis (e.g., **`getAinc()`**), some of the arrays are *vestigial*; that is, they are dimensioned as scalars. These arrays are: **`psl_sigF(1,1,1)`**, **`usl_sigF(1,1,1)`**, **`vsl_sigF(1,1,1)`**, **`mix_sigF(1,1,1)`**, **`psl_inc(1,1,1)`**, **`usl_inc(1,1,1)`**, **`vsl_inc(1,1,1)`**, **`mix_inc(1,1,1)`**, **`psl_sigA(1,1,1)`**, **`usl_sigA(1,1,1)`**, **`vsl_sigA(1,1,1)`**, and **`mix_sigA(1,1,1)`**.
- The **`LOGICAL`** parameters **`want_psl`**, **`want_usl`**, **`want_vsl`**, and **`want_mix`** are set to **`.FALSE.`**
- Since no upper-air AI's are to be calculated, calls to the routines **`dervsigF_slD()`** and **`dervsigF_slW()`** are not made.

Table 37: Data passed into `getAIzuv()` via its interface.

Variable	Type	Intent	Description
npieces	INTEGER	INOUT	Number of Innovations
lat_list	REAL(npieces)	INOUT	Latitude
lon_list	REAL(npieces)	INOUT	Longitude
pres_list	REAL(npieces)	INOUT	Pressure (hPa)
time_list	REAL(npieces)	INOUT	Δt (min) from Analysis Time
kx_list	INTEGER(npieces)	INOUT	Data Source Index **kx**
kt_list	INTEGER(npieces)	INOUT	Data Type Index **kt**
dels_list	REAL(npieces)	INOUT	Innovation O-F
sigF_list	REAL(npieces)	INOUT	Forecast Error Variances σ_f
sigO_list	REAL(npieces)	INOUT	Observation Error Variances σ_o
im	INTEGER	IN	Number of Longitudinal Grid Points
jnp	INTEGER	IN	Number of Latitudinal Grid Points
mlev	INTEGER	IN	Number of Grid Vertical Levels
pres_lev	REAL(mlev)	INOUT	Pressure Levels
z_sigF	REAL(im,jnp)	IN	Forecast Geopotential Height Error Variances
u_sigF	REAL(im,jnp)	IN	Forecast u Error Variances
v_sigF	REAL(im,jnp)	IN	Forecast v Error Variances
z_inc	REAL(im,jnp)	OUT	Geopotential Height Analysis Increments
u_inc	REAL(im,jnp)	OUT	u Analysis Increments
v_inc	REAL(im,jnp)	OUT	v Analysis Increments
z_sigA	REAL(im,jnp)	OUT	Geopotential Height Analysis Error Variances
u_sigA	REAL(im,jnp)	OUT	u Analysis Error Variances
v_sigA	REAL(im,jnp)	OUT	v Analysis Error Variances

The Upper-air Water Vapor Mixing Ratio Analysis `getAImix()` The PSAS interface that computes only water vapor mixing ratio analysis increments is the routine **`getAImix()`**, whose interface is shown below.

```
subroutine getAImix( npieces, lat_list, lon_list, pres_list, &
                     time_list, kx_list, kt_list, dels_list, &
                     sigF_list, sigO_list,                   &
                     im, jnp, mlev, pres_lev,                &
                     mix_sigF, mix_inc, mix_sigA              )
```

The calling tree for **`getAImix()`** is shown in Figure 9, and its arguments are summarized in Table 38.

This routine is modified from **`getAIall()`** in the following way:

- In order to keep the same interface to some routines called in the analysis (e.g., **`getAinc()`**), some of the arrays are *vestigial*; that is, they are dimensioned as scalars. These arrays are: `psl_sigF(1,1,1)`, `usl_sigF(1,1,1)`, `vsl_sigF(1,1,1)`, `z_sigF(1,1,1)`, `u_sigF(1,1,1)`, `v_sigF(1,1,1)`, `psl_inc(1,1,1)`, `usl_inc(1,1,1)`, `vsl_inc(1,1,1)`, `z_inc(1,1,1)`, `u_inc(1,1,1)`, `v_inc(1,1,1)`, `psl_sigA(1,1,1)`, `usl_sigA(1,1,1)`, `vsl_sigA(1,1,1)`, `z_sigA(1,1,1)`, `u_sigA(1,1,1)`, and `v_sigA(1,1,1)`.
- The `LOGICAL` parameters `want_psl`, `want_usl`, `want_vsl`, `want_z`, `want_u`, and `want_v` are set to `.FALSE.`
- Since no multivariate upper-air AI's are to be calculated, calls to the routines **`dervsigF_slD()`**, **`dervsigF_slW()`**, **`dervsigF_upD()`**, and **`dervsigF_upW()`** are not made.

Table 38: Data passed into `getAImix()` via its interface.

Variable	Type	Intent	Description
npieces	INTEGER	INOUT	Number of Innovations
lat_list	REAL(npieces)	INOUT	Latitude
lon_list	REAL(npieces)	INOUT	Longitude
pres_list	REAL(npieces)	INOUT	Pressure (hPa)
time_list	REAL(npieces)	INOUT	Δt (min) from Analysis Time
kx_list	INTEGER(npieces)	INOUT	Data Source Index **kx**
kt_list	INTEGER(npieces)	INOUT	Data Type Index **kt**
dels_list	REAL(npieces)	INOUT	Innovation O-F
sigF_list	REAL(npieces)	INOUT	Forecast Error Variances σ_f
sigO_list	REAL(npieces)	INOUT	Observation Error Variances σ_o
im	INTEGER	IN	Number of Longitudinal Grid Points
jnp	INTEGER	IN	Number of Latitudinal Grid Points
mlev	INTEGER	IN	Number of Grid Vertical Levels
pres_lev	REAL(mlev)	INOUT	Pressure Levels
mix_sigF	REAL(im,jnp)	IN	Forecast Mixing Ratio Error Variances
mix_inc	REAL(im,jnp)	OUT	Mixing Ratio Analysis Increments
mix_sigA	REAL(im,jnp)	OUT	Mixing Ratio Analysis Error Variances

Appendix B: Complete Procedural Flow Diagram

Here we give a complete control flow diagram for the static test driver program **psasshell.F** and trace its call activity through the analysis interface **getAIall()**. The diagram was generated from PSAS source code using the tools **Fent** and **Ftr**, both of which were created by Jing Guo of the NASA DAO.

```
(psasshell.F):
   |-luavail(luavail.f)
   |-lnblnk(lnblnk.f)
   |-pZEITBEG(zeits.F)
   .  |-syszeit(zeits.F)
   .  .  |-second(?)
   .  .  |-secondr(?)
   .  .  |-time(?)
   .  .  |-etime(?)
   .  |-syszeit(^)
   |-getenv(?)
   |-I90_LoadF(m_inpak90.f90)
   .  |-opntext(opntext.F)
   .  .  |-asnunit(?)
   .  |-i90_page(m_inpak90.f90)
   .  |-i90_trim(m_inpak90.f90)
   .  |-i90_pad(m_inpak90.f90)
   |-psasexit(psasexit.f)
   .  |-pzeitpri(zeits.F)
   .  .  |-syszeit(^)
   .  |-exit(?)
   |-pZEITBEG(^)
   |-GETDEL2(getdel2.f)
   .  |-luavail(^)
   .  |-lnblnk(^)
   .  |-LABLIN(m_inpak90.f90)
   .  .  |-i90_label(m_inpak90.f90)
   .  |-STRGET(m_inpak90.f90)
   .  .  |-i90_gstr(m_inpak90.f90)
   .  .  .  |-i90_trim(^)
   .  |-opnieee(opnieee.F)
   .  .  |-asnunit(^)
   |-pZEITEND(zeits.F)
   .  |-syszeit(^)
   |-psasexit(^)
   |-iniainc(iniainc.f)
   .  |-LABLIN(^)
   |-iniaio(aio_grads.f)
   .  |-LUAVAIL(^)
   .  |-amatch(amatch.f)
   .  .  |-lnblnk(^)
   .  |-listvals(listvals.f)
   .  |-psasexit(^)
   .  |-LABLIN(^)
   .  |-STRGET(^)
   .  |-psasexit(^)
   .  |-lablin(^)
   .  |-LABLIN(^)
   .  |-rdlevels(rdlevels.f)
   .  .  |-lnblnk(^)
   .  .  |-lablin(^)
```

```
. . |-getwrd(m_inpak90.f90)
. . . |-i90_gtoken(m_inpak90.f90)
. . . . |-i90_trim(^)
. . |-getwrd(^)
. |-LABLIN(^)
. |-psasexit(^)
. |-LABLIN(^)
. |-psasexit(^)
. |-LABLIN(^)
. |-psasexit(^)
. |-opnieee(^)
. |-psasexit(^)
|-psasexit(^)
|-I90_Release(m_inpak90.f90)
|-pzeitbeg(^)
|-getAIall(getAIall.F)
. |-snrm2(?)
. |-luavail(^)
. |-lnblnk(^)
. |-psasrcbd(psasrcbd.f)
. |-pzeitbeg(^)
. |-psasexit(^)
. |-getenv(^)
. |-pzeitbeg(^)
. |-I90_LoadF(^)
. |-pzeitend(^)
. |-psasexit(^)
. |-pzeitbeg(^)
. |-initRSRC(initRSRC.F90)
. . |-qtrig0(qtrig0.f)
. . |-ktname0(ktname0.f)
. . . |-rdkttbl(rdkttbl.f)
. . . . |-lnblnk(^)
. . . . |-lablin(^)
. . . . |-rdnext(m_inpak90.f90)
. . . . . |-i90_gline(m_inpak90.f90)
. . . . |-getwrd(^)
. . . . |-getstr(m_inpak90.f90)
. . . . . |-i90_gstr(^)
. . . . |-getwrd(^)
. . . . |-rdnext(^)
. . . |-psasexit(^)
. . |-kxname0(kxname0.f)
. . . |-rdkxtbl(rdkxtbl.f)
. . . . |-lnblnk(^)
. . . . |-lablin(^)
. . . . |-rdnext(^)
. . . . |-getwrd(^)
. . . . |-getstr(^)
. . . . |-rdnext(^)
. . . |-psasexit(^)
. . |-set_OEclas(set_OEclas.F90)
. . . |-listvals(^)
. . . |-lstins(lstins.f)
. . . . |-lnblnk(^)
. . . |-tabSlist(tabSlist.f90)
. . . . |-psasexit(^)
. . . . |-indexxs(indexxs.f90)
. . . . |-getTab_(tabIlist.F90)
```

```
. . . . . |-nextTab_(tabIlist.F90)
. . . . . |-getVal_(tabIlist.F90)
. . . . . |-nextVal_(tabIlist.F90)
. . . . . |-getTab_(^)
. . . . . |-nextTab_(^)
. . . . . |-nextVal_(^)
. . . . . |-getTab_(^)
. . . . . |-nextTab_(^)
. . . . . |-getVal_(^)
. . . . . |-nextVal_(^)
. . . . |-psasexit(^)
. . . . |-inxSlist(inxSlist.f90)
. . . . |-rdlevels(^)
. . . . |-psasexit(^)
. . . . |-rdoetbl(rdoetbl.f)
. . . . . |-lnblnk(^)
. . . . . |-lablin(^)
. . . . . |-rdnext(^)
. . . . . |-getwrd(^)
. . . . . |-rdnext(^)
. . . . |-psasexit(^)
. . . |-set_OEhCor(set_OEhCor.F90)
. . . . |-tabSlist(^)
. . . . |-inxSlist(^)
. . . . |-rdpars(rdpars.f90)
. . . . . |-lablin(^)
. . . . . |-rdnext(^)
. . . . . |-getwrd(^)
. . . . . |-getstr(^)
. . . . . |-getwrd(^)
. . . . . |-rdnext(^)
. . . . |-psasexit(^)
. . . |-set_OEvCor(set_OEvCor.f90)
. . . . |-tabSlist(^)
. . . . |-inxSlist(^)
. . . . |-rdvctbl(rdvctbl.f90)
. . . . . |-lnblnk(^)
. . . . . |-lablin(^)
. . . . . |-rdnext(^)
. . . . . |-getwrd(^)
. . . . . |-getstr(^)
. . . . . |-getwrd(^)
. . . . . |-rdnext(^)
. . . . |-psasexit(^)
. . . |-set_FEhCor(set_FEhCor.f90)
. . . . |-rdpars(^)
. . . . |-psasexit(^)
. . . . |-rdpars(^)
. . . . |-psasexit(^)
. . . |-set_FEvCor(set_FEvCor.f90)
. . . . |-rdvctbl(^)
. . . . |-psasexit(^)
. . . . |-rdvctbl(^)
. . . . |-psasexit(^)
. . . |-tabl_FEsigW(tabl_FEsigW.f90)
. . . . |-rdpars(^)
. . . . |-psasexit(^)
. . . |-tabl_FEalpha(tabl_FEalpha.f90)
. . . . |-rdPars(^)
```

```
. . . |-psasexit(^)
. . |-bands0(bands0.f)
. . . |-LABLIN(^)
. . . |-exit(^)
. . . |-LABLIN(^)
. . . |-exit(^)
. . . |-LABLIN(^)
. . . |-lablin(^)
. . . |-psasexit(^)
. . |-krname0(krname0.f)
. . |-seticos(seticos.f)
. . |-PROXEL0(proxel.f)
. . . |-LABLIN(^)
. . |-iniAInc(^)
. |-pzeitend(^)
. |-intp_sig0(intp_sig0.F90)
. . |-listvals(^)
. . |-psasexit(^)
. . |-slogtab(slogtab.f90)
. . . |-vindex(vindex.f)
. |-obstat(obstat.f)
. . |-lnblnk(^)
. |-obstat(^)
. |-PARDISP(pardisp.f)
. |-OBSSMRY(obssmry.f)
. |-pzeitbeg(^)
. |-restrict(restrict.F)
. . |-setbox(setbox.f90)
. . . |-lablin(^)
. . . |-rdnext(^)
. . . |-psasexit(^)
. . . |-getwrd(^)
. . . |-psasexit(^)
. . . |-getwrd(^)
. . . |-psasexit(^)
. . . |-rdnext(^)
. . . |-psasexit(^)
. . |-psasexit(^)
. . |-INITKL(initkl.f)
. . |-LLBOXES(llboxes.f)
. . |-mark_nsig(mark_nsig.f90)
. . |-TOFRONT(tofront.F)
. . . |-psasexit(^)
. . . |-INDEXXR(indexxr.f)
. . . |-PERMUTI(permuti.f)
. . . |-PERMUTI(^)
. . . |-PERMUTL(permutl.f)
. . . |-PERMUTR(permutr.f)
. . . |-PERMUTR(^)
. . |-PRTDATA(prtdata.f)
. |-pzeitend(^)
. |-pzeitbeg(^)
. |-sort(sort.f)
. . |-regsort(regsort.F)
. . . |-psasexit(^)
. . . |-LL2XYZ(ll2xyz.F)
. . . . |-qtrig(qtrig.f)
. . . . |-qtrig(^)
. . . |-XYZ2REG(xyz2reg.f)
```

```
. . . . |-ipick(ipick.f)
. . . |-INDEXXI(indexxi.f)
. . . |-PERMUTI(^)
. . . |-PERMUTR(^)
. . |-psasexit(^)
. . |-typsort(typsort.F)
. . . |-psasexit(^)
. . . |-INDEXXI(^)
. . . |-PERMUTI(^)
. . . |-PERMUTR(^)
. . . |-INDEX3R(index3r.f)
. . . |-PERMUTI(^)
. . . |-PERMUTR(^)
. |-pzeitend(^)
. |-pzeitbeg(^)
. |-dupelim(dupelim.f)
. . |-TOFRONT(^)
. |-pzeitend(^)
. |-pzeitbeg(^)
. |-proxel(proxel.f)
. . |-sepang(sepang.f)
. . |-psasexit(^)
. . |-LL2XYZ(^)
. . |-psasexit(^)
. . |-PRXSOB(proxel.f)
. . |-psasexit(^)
. . |-TOFRONT(^)
. |-pzeitend(^)
. |-gridxx0(gridxx.f)
. . |-LABLIN(^)
. |-OBSSMRY(^)
. |-obstat(^)
. |-pZEITBEG(^)
. |-merg_plevs(merg_plevs.F90)
. . |-psasexit(^)
. . |-indexxi(^)
. . |-permuti(^)
. |-merg_plevs(^)
. |-merg_lats(merg_lats.F90)
. . |-psasexit(^)
. . |-tabRlist(tabRlist.F90)
. . . |-psasexit(^)
. . . |-tabIlist(tabIlist.F90)
. . . . |-psasexit(^)
. . . . |-indexxi(^)
. . . . |-getTab_(^)
. . . . |-nextTab_(^)
. . . . |-getVal_(^)
. . . . |-nextVal_(^)
. . . . |-getTab_(^)
. . . . |-nextTab_(^)
. . . . |-nextVal_(^)
. . . . |-getTab_(^)
. . . . |-nextTab_(^)
. . . . |-getVal_(^)
. . . . |-nextVal_(^)
. . |-tabRlist(^)
. |-set_fecHH(set_fecHH.F90)
. . |-intp_ctaus(intp_ctaus.F90)
```

```
. . . |-psasexit(^)
. . |-psasexit(^)
. . |-intp_hCor(intp_hCor.F90)
. . . |-psasexit(^)
. . . |-slogtab(^)
. . . |-corfun(hcorfuns.F90)
. . . . |-damp_cosine(hcorfuns.F90)
. . . . |-gaussian(hcorfuns.F90)
. . . . |-exponential(hcorfuns.F90)
. . . . |-power_law(hcorfuns.F90)
. . . . |-gaspari_cohn(hcorfuns.F90)
. . . . |-win_powerlaw_(hcorfuns.F90)
. . . |-psasexit(^)
. . |-intp_vCor(intp_vCor.F90)
. . . |-psasexit(^)
. . . |-slogtab(^)
. |-set_oecHH(set_oecHH.F90)
. . |-intp_ctaus(^)
. . |-intp_hCor(^)
. . |-intp_vCor(^)
. |-set_fecQQ(set_fecQQ.F90)
. . |-intp_ctaus(^)
. . |-intp_hCor(^)
. . |-intp_vCor(^)
. |-imat_alpha(imat_alpha.F90)
. . |-psasexit(^)
. . |-EXP_ALPHAS_model_x1(imat_alpha.F90)
. . . |-psasexit(^)
. . . |-slogtab(^)
. . |-psasexit(^)
. . |-EXP_ALPHAS_model_(imat_alpha.F90)
. . . |-psasexit(^)
. . . |-slogtab(^)
. . |-psasexit(^)
. . |-EPSILONS_model_(imat_alpha.F90)
. . . |-psasexit(^)
. . . |-slogtab(^)
. . |-OI_model_(imat_alpha.F90)
. . |-psasexit(^)
. |-imat_sigW(imat_sigW.F90)
. . |-psasexit(^)
. . |-PSAS11_sigW_a(imat_sigW.F90)
. . . |-psasexit(^)
. . . |-slogtab(^)
. . |-PSAS11_sigW_(imat_sigW.F90)
. . . |-psasexit(^)
. . . |-slogtab(^)
. . |-OI_sigW_(imat_sigW.F90)
. . . |-psasexit(^)
. . . |-slogtab(^)
. . |-psasexit(^)
. . |-slogtab(^)
. |-pzeitend(^)
. |-pzeitbeg(^)
. |-getsigF(getsigF.f)
. . |-LUAVAIL(^)
. . |-lnblnk(^)
. . |-LABLIN(^)
. . |-STRGET(^)
```

```
. . |-I90_Release(^)
. . |-psasexit(^)
. . |-GRADS0(m_grads.f90)
. . . |-luavail(^)
. . . |-i90_loadf(^)
. . . |-i90_label(^)
. . . |-i90_gtoken(^)
. . . |-i90_label(^)
. . . |-i90_gtoken(^)
. . . |-i90_label(^)
. . . |-i90_gline(^)
. . . |-i90_gtoken(^)
. . . |-i90_label(^)
. . . |-i90_gtoken(^)
. . . |-i90_release(^)
. . . |-opnieee(^)
. . |-psasexit(^)
. . |-RGRADS(m_grads.f90)
. . |-psasexit(^)
. . |-RGRADS(^)
. . |-dervsigF_slp(dervsigF_slp.F90)
. . |-RGRADS(^)
. . |-psasexit(^)
. . |-RGRADS(^)
. . |-psasexit(^)
. . |-lvstat(lvstat.f)
. . |-GDSTAT(gdstat.f)
. . |-GDSTAT(^)
. . |-GRADS1(m_grads.f90)
. |-pzeitend(^)
. |-pzeitbeg(^)
. |-dervsigF_slD(dervsigF_slD.F90)
. . |-slogtab(^)
. |-pzeitend(^)
. |-lvstat(^)
. |-pzeitbeg(^)
. |-dervsigF_upD(dervsigF_upD.F90)
. . |-psasexit(^)
. . |-slogtab(^)
. |-pzeitend(^)
. |-GDSTAT(^)
. |-pzeitbeg(^)
. |-intp_sigF(intp_sigF.F90)
. . |-psasexit(^)
. . |-slogtab(^)
. |-pzeitend(^)
. |-obstat(^)
. |-pZEITBEG(^)
. |-psasexit(^)
. |-makedelv(makedelv.F)
. . |-psasexit(^)
. . |-ll2qvec(ll2qvec.f90)
. . |-setpix(setpix.f90)
. . |-slogtab(^)
. . |-slintab(slintab.f90)
. . . |-vindex(^)
. . |-op_Mx(op_Mx.F90)
. . . |-sparse(sparse.f90)
. . . . |-sepang(^)
```

```
. . . |-snrm2(^)
. . . |-psasexit(^)
. . . |-covOOxpy_(op_Mx.F90)
. . . .  |-mv_diag(mv_diag.f90)
. . . .  .  |-psasexit(^)
. . . .  |-sym_Cxpy(sym_Cxpy.F90)
. . . .  .  |-sparse(^)
. . . .  .  |-snrm2(^)
. . . .  .  |-pzeitbeg(^)
. . . .  .  |-pzeitend(^)
. . . .  .  |-Cprod1(Cprod1.F90)
. . . .  .  .  |-diagcorF(cordriv.f90)
. . . .  .  .  |-diagcor0(cordriv.f90)
. . . .  .  .  |-diagcorU(cordriv.f90)
. . . .  .  .  |-SSPMV(?)
. . . .  .  |-pzeitend(^)
. . . .  .  |-Cprodx(Cprodm.f90)
. . . .  .  |-pzeitend(^)
. . . .  |-psasexit(^)
. . . .  |-mv_diag(^)
. . . .  |-sym_Cxpy(^)
. . . .  |-psasexit(^)
. . . .  |-mv_diag(^)
. . . |-covFFxpy_(op_Mx.F90)
. . . .  |-aj_Alf(aj_Alf.f90)
. . . .  .  |-psasexit(^)
. . . .  |-mv_diag(^)
. . . .  |-sym_Cxpy(^)
. . . .  |-psasexit(^)
. . . .  |-mv_diag(^)
. . . .  |-mv_Alf(mv_Alf.f90)
. . . .  .  |-psasexit(^)
. . . .  |-getivec(getivec.F90)
. . . .  |-aj_Bet(aj_Bet.f90)
. . . .  .  |-psasexit(^)
. . . .  |-mv_diag(^)
. . . .  |-sym_Cxpy(^)
. . . .  |-psasexit(^)
. . . .  |-mv_diag(^)
. . . .  |-mv_Bet(mv_Bet.f90)
. . . .  .  |-psasexit(^)
. . . .  |-getivec(^)
. . . .  |-aj_Gam(aj_Gam.f90)
. . . .  .  |-psasexit(^)
. . . .  |-mv_diag(^)
. . . .  |-sym_Cxpy(^)
. . . .  |-psasexit(^)
. . . .  |-mv_diag(^)
. . . .  |-mv_Gam(mv_Gam.f90)
. . . .  .  |-psasexit(^)
. . . |-mv_diag(^)
. . . |-sym_Cxpy(^)
. . . |-psasexit(^)
. . . |-mv_diag(^)
. . . |-sym_Cxpy(^)
. . . |-psasexit(^)
. . . |-mv_diag(^)
. . |-obstat(^)
. . |-solve4x(solve4x.F)
```

```
. . |-snrm2(^)
. . |-psasexit(^)
. . |-ll2qvec(^)
. . |-setpix(^)
. . |-slogtab(^)
. . |-slintab(^)
. . |-conjgr(conjgr.F)
. . . |-sdot(?)
. . . |-snrm2(^)
. . . |-pzeitbeg(^)
. . . |-SCOPY(?)
. . . |-SCOPY(^)
. . . |-conjgr2(conjgr.F)
. . . . |-sparse(^)
. . . . |-sdot(^)
. . . . |-snrm2(^)
. . . . |-pzeitbeg(^)
. . . . |-SCOPY(^)
. . . . |-pzeitbeg(^)
. . . . |-conjgr1(conjgr.F)
. . . . . |-sdot(^)
. . . . . |-snrm2(^)
. . . . . |-lnblnk(^)
. . . . . |-luavail(^)
. . . . . |-SCOPY(^)
. . . . . |-diagcorM(diagcorM.F90)
. . . . . |-psasexit(^)
. . . . . |-diagcor0(^)
. . . . . |-check_istat_(diagcorM.F90)
. . . . . |-add_corM_(diagcorM.F90)
. . . . . |-add_smat_(diagcorM.F90)
. . . . . |-add_iden_(diagcorM.F90)
. . . . . |-add_diag_(diagcorM.F90)
. . . . . |-diagcorU(^)
. . . . . |-check_istat_(^)
. . . . . |-add_corM_(^)
. . . . . |-diagcorD(diagcorD.f90)
. . . . . |-fDDcor1(cormats.f90)
. . . . . |-fDDcorx(cormats.f90)
. . . . . |-fDDcor1(^)
. . . . . |-diagcorF(^)
. . . . . |-check_istat_(^)
. . . . . |-add_corM_(^)
. . . . . |-diagcorD(^)
. . . . . |-check_istat_(^)
. . . . . |-add_corM_(^)
. . . . . |-diagcorD(^)
. . . . . |-check_istat_(^)
. . . . . |-add_corM_(^)
. . . . . |-SCOPY(^)
. . . . . |-smex(smex.f)
. . . . . |-SPPTRF(?)
. . . . . |-SPPTRS(?)
. . . . . |-SCOPY(^)
. . . . . |-SAXPY(?)
. . . . . |-SCOPY(^)
. . . . . |-SSPMV(^)
. . . . . |-SAXPY(^)
. . . . . |-cgnorm(cgnorm.f)
```

```
. . . . . . |-lnblnk(^)
. . . . . |-pzeitend(^)
. . . . . |-SCOPY(^)
. . . . . |-SAXPY(^)
. . . . . |-SCOPY(^)
. . . . . |-op_Mx(^)
. . . . . |-SAXPY(^)
. . . . . |-cgnorm(^)
. . . . . |-pzeitend(^)
. . . . |-pzeitend(^)
. . . . |-SCOPY(^)
. . . . |-SAXPY(^)
. . . . |-SCOPY(^)
. . . . |-op_Mx(^)
. . . . |-SAXPY(^)
. . . . |-cgnorm(^)
. . . . |-pzeitend(^)
. . . |-psasexit(^)
. . . |-conjgr(^)
. . . |-psasexit(^)
. . |-pZEITEND(^)
. . |-obstat(^)
. . |-pzeitbeg(^)
. . |-getAinc(getAinc.F)
. . . |-gridxx(gridxx.f)
. . . . |-EAGRID(gridxx.f)
. . . . |-LL2XYZ(^)
. . . . |-XYZ2REG(^)
. . . . |-INDEXXI(^)
. . . . |-PERMUTR(^)
. . . |-intp_sigF(^)
. . . |-mvPHx(mvPHx.F)
. . . . |-snrm2(^)
. . . . |-psasexit(^)
. . . . |-ll2qvec(^)
. . . . |-slintab(^)
. . . . |-indexGvec(indexGvec.F90)
. . . . . |-psasexit(^)
. . . . |-psasexit(^)
. . . . |-setpindx(setpindx.f)
. . . . . |-hunt(hunt.f)
. . . . . . |-psasexit(^)
. . . . |-setpindx(^)
. . . . |-ll2qvec(^)
. . . . |-slintab(^)
. . . . |-slogtab(^)
. . . . |-psasexit(^)
. . . . |-op_Pf(op_Pf.F90)
. . . . . |-sparse(^)
. . . . . |-snrm2(^)
. . . . . |-psasexit(^)
. . . . . |-covFFxpy_(^)
. . . . . |-psasexit(^)
. . . . . |-pzeitbeg(^)
. . . . . |-aj_Alf(^)
. . . . . |-mv_diag(^)
. . . . . |-rec_Cxpy(rec_Cxpy.F90)
. . . . . . |-sparse(^)
. . . . . . |-snrm2(^)
```

```
. . . . . |-gCprodx(Cprodm.f90)
. . . . |-psasexit(^)
. . . . |-mv_diag(^)
. . . . |-mv_Alf(^)
. . . . |-pzeitend(^)
. . . . |-pzeitbeg(^)
. . . . |-getivec(^)
. . . . |-aj_Bet(^)
. . . . |-mv_diag(^)
. . . . |-rec_Cxpy(^)
. . . . |-psasexit(^)
. . . . |-mv_diag(^)
. . . . |-mv_Bet(^)
. . . . |-pzeitend(^)
. . . . |-pzeitbeg(^)
. . . . |-getivec(^)
. . . . |-aj_Gam(^)
. . . . |-mv_diag(^)
. . . . |-rec_Cxpy(^)
. . . . |-psasexit(^)
. . . . |-mv_diag(^)
. . . . |-mv_Gam(^)
. . . . |-pzeitend(^)
. . |-permutr(^)
. . |-Gea2ll(Gea2ll.F)
. . . |-ea2ll(gridxx.f)
. . . . |-psasexit(^)
. . . . |-SPLINE(spline.f)
. . . . |-SPLINE(^)
. . . |-ea2ll(^)
. |-psasexit(^)
. |-lvstat(^)
. |-gdstat(^)
. |-pzeitend(^)
. |-pzeitbeg(^)
. |-dervsigF_slW(dervsigF_slW.F90)
. . |-psasexit(^)
. . |-slogtab(^)
. . |-slintab(^)
. |-pzeitend(^)
. |-lvstat(^)
. |-pzeitbeg(^)
. |-dervsigF_upW(dervsigF_upW.F90)
. . |-psasexit(^)
. . |-slogtab(^)
. . |-slintab(^)
. |-pzeitend(^)
. |-GDSTAT(^)
. |-pzeitbeg(^)
. |-intp_sigF(^)
. |-pzeitend(^)
. |-obstat(^)
. |-pzeitend(^)
|-pzeitend(^)
|-pzeitbeg(^)
|-opnieee(^)
|-psasexit(^)
|-wrtxvec(wrtxvec.f)
|-psasexit(^)
```

```
|-pzeitend(^)
|-pzeitbeg(^)
|-wrtaio(aio_grads.f)
.  |-psasexit(^)
.  |-WTBUF(aio_grads.f)
.  |-psasexit(^)
.  |-WTBUF(^)
.  |-psasexit(^)
.  |-WTBUF(^)
.  |-psasexit(^)
.  |-WTBUF(^)
.  |-psasexit(^)
.  |-WTBUF(^)
.  |-psasexit(^)
.  |-WTBUF(^)
.  |-psasexit(^)
.  |-WTBUF(^)
.  |-psasexit(^)
|-wrtaio(^)
|-pzeitend(^)
|-endaio(aio_grads.f)
.  |-psasexit(^)
.  |-opntext(^)
|-pZEITEND(^)
|-psasexit(^)
```

Appendix C: A Sample Resource File psas.rc

```
# PSAS.header.rsrc
#-------------------------------------------------------------------------
#/////////////////////////////////////////////////////////////////////////
#-------------------------------------------------------------------------
#
# A little bit of the history:
#
# 01Oct96  - J. Guo - commited version 103 with OI2.5 alike data
# 15Aug96b - J. Guo - commited version 102 with modified correlation
#   function for FcstErr and ObsErr for tovA in
#   v102/.
# 07Aug96b - J. Guo - commited version 101
# 07Aug96a - J. Guo - commited version 100
# 07Aug96 - J. Guo - Put under CVS control
# 21Hay96 - J. Guo - Included tuned statistics of v100.
#
# 05Oct95 - Jing G. - New vertical correlation functions.  Earlier
#   tables are not positive definite.
#
#   + FcstErr*vCor_HH:: the 14 level table was PD,
#     but not the 18 level table;
#
#   + ObsErr*vCor_HH-[134]:: tables in for both 14
#     and 18 levels are either not PD or poorly
#     conditioned.
#
# - Added comments for some earlier changes:
#
#   + Precondition conjgr1() convergence rate
#   + New entries
#
# 13Jun95 - Jing G. - Fixed an error in the earlier version with
#   all data above 30 mb excluded.
# 31Har95 - Jing G. - Created a version with GEOS-1.0 and original
#   PSAS, with 18 levels error statistics.
# 02Feb95 - Jing G. - Added time box for setbox()
# - Edited for generic libpsas.a calls.
# - Removed some blanks in one of entry, "del_log_
#   for...:", both in this file and in proxel().
#   It may cause some backward compatibility prog-
#   ram.  Since this is the first version to be
#   compiled for libpsas.a, it is better to make
#   it right.
# 06Jan95 - Jing G. - Added kx-kt boxes
# 30Dec94 - Jing G. - Removed Analysis-Increment-Computation-
#   Restrictions.
# - Haking a test for sealevel wind analysis
# file: psas.rc - last change: 01dec94 Jing G. - added option to
#   added option to restrict data included in aio
#   computations.
# file: psas.rc - last change: 04oct94 (A. da S.)
#
#   This is an INPAK77 style resource file for input of PSAS parameters.
#
#-------------------------------------------------------------------------
#/////////////////////////////////////////////////////////////////////////
```

```
#-----------------------------------------------------------------------
#.

# PSAS.test.rsrc

#                INPUT/OUTPUT FILE NAMES
#                     ----------------------

innovation_file_name:                ../../data/oo.del2.p500.n00.xu $
analysis_increment_output_file_name: _EXPID_.d_DATE_.t_TIME_.grd  $

fcst_err_grads_descr_file: <>

# Only inquired when upper air variable analysis is wanted
# ========================================================
want_analysis_incs_only_on_selected_levels:     yes

# Use default control if the answer is "no"
#==========================================
pressure_interval_for_analysis_incs:          49 851

# Use following entries if the answer is "yes"
#=============================================

# 29 levels

#Analysis_Incs_Levels: Pressure 1000 925 850 700 500 400 300 250 200 150 100 70 50 40 30 20 15 10 7 5 3 2 1 .7 .5 .4 .2 .05 .01

# 18 levels

Analysis_Incs_Levels: Pressure .4 1 2 5 10 30 50 70 100 150 200 250 300 400 500 700 850 1000

# Testing levels
#Analysis_Incs_Levels: Pressure .01 10 200 500
#Analysis_Incs_Levels: Pressure 500

#                     ANALYSIS INCREMENT OUTPUT
#                     -------------------------

want_analysis_incs_of_sealevel_u:            yes      # yes or no
want_analysis_incs_of_sealevel_v:            yes      # yes or no
want_analysis_incs_of_sealevel_pressure:     yes      # yes or no

want_analysis_incs_of_upper_level_u:         yes       # yes or no
want_analysis_incs_of_upper_level_v:         yes       # yes or no
want_analysis_incs_of_upper_level_heights:   yes       # yes or no
want_analysis_incs_of_upper_level_mixr:      yes      # yes or no
#.

# PSAS.run.rsrc

#-----------------------------------------------------------------------
#          Equal-Area intermediate grid settings

latitude_treshold_for_equal_area_grid:       45

#-----------------------------------------------------------------------
#                     PROXIMITY CHECK PARAMETERS
#                     --------------------------

radius_in_km_for_proximity_check:       300.              # Rkm
```

```
del_lnp_for_vertical_proximity_check:   0.1              # dellnp
do_you_want_detailed_superob_list:       n               # y or n

#--------------------------------------------------------------------
#                   CONJUGATE GRADIENT PARAMETERS
#                   -----------------------------

# The next entry should be removed
method_of_solution:                     1   # 0 direct solver
                                            # 1 conjugate gradient

level_for_banded_approximation:         5   # 4 neglect corr R > 3000 km
                                            # 5 neglect corr R > 6000 km
                                            # (actual distances depend
                                            #  on seplim below).

# The next entry should be removed.
frequency_to_output_analysis_incs_during_cg:  999 # 1 every iteration,
                                                  # 2 every other
                                                  # 9999 never


#             Levels (*) -->     1      2    3    4      5
#                               -----   ---  ---  ----   ----
conjgr_separation_limits:       0       0    0    26.5   58.25  # seplim
conjgr_maximum_no_iterations:   32      16   1    8      8      # maxpass
conjgr_minimum_no_iterations:   16      8    1    4      4      # minpass
conjgr_max_tolerances:          0.1     0.1  0.1  0.1    0.1    # criter
conjgr_min_tolerances:          0.3     0.3  0.3  0.3    0.3    # criter
conjgr_tolerances:              0.1     0.1  0.1  0.1    0.1    # criter

conjgr_H_small:                 16                               # msmall

conjgr_verbose:         0       1    0    1      1      # 0 = F
       # 1 = T

matrix_conditioning_factor:     1.        # 1.

# (*) Level 3 is obsolete; set level 4 or 5 used according to your choice
#     of level_for_banded_approximation above. All levels must be set
#     even if not used.

# matrix_conditioning_factor: may be removed if good P.D correlation
# matrices are expected.
#.

DataBoxes::

#--------------------------------------------------------------------
#       NASA/GSFC, Data Assimilation Office, Code 910.3, GEOS/DAS      !
#--------------------------------------------------------------------
#
# !NAME: DataBoxes:: - Data selection tables in 6-d boxes
#
# !REVISION HISTORY:
#  09May96 - J. Guo - implemented to remplace the original
# PSAS "regional boxes" resources.
#--------------------------------------------------------------------

#   kx   kt   lat     lon   pres time
#--------------------------------------------------------------------
```

```
  1  32   1  7  -90 +90  -180 +180     .009 1000.1   -180 +180

#-------------------------------------------------------------------------
# Levels interpolated from NHC/NESDIS-TOVS removed

# a list of TOVS' levels are removed.  The remained levels
# include

# .4
# 1
# 2

 33  56   1  7  -90 +90  -180 +180     .39     2.1   -180 +180

#   not 3
# 5

 33  56   1  7  -90 +90  -180 +180    4.9      5.1   -180 +180

#   not 7
# 10

 33  56   1  7  -90 +90  -180 +180    9.9     10.1   -180 +180

#   not 20
#       30
# 50
# 70
# 100

 33  56   1  7  -90 +90  -180 +180   29.9    100.1   -180 +180

#   not 150
# 200

 33  56   1  7  -90 +90  -180 +180  199.9    200.1   -180 +180

#   not 250
# 300
# 400
# 500
# 700
# 850

 33  56   1  7  -90 +90  -180 +180  299.9    850.1   -180 +180
#-------------------------------------------------------------------------

 57  87   1  7  -90 +90  -180 +180     .009 1000.1   -180 +180

#-------------------------------------------------------------------------
#   kx   kt   lat     lon   pres time

::

# DataTypeTable.rsrc
#-------------------------------------------------------------------------
#//////////////////////////////////////////////////////////////////////////
#-------------------------------------------------------------------------
#
# 09Jan95 - Jing G. - Tabulated version of "kt"
```

```
#
#      kt : Original assigned sequence numbers, integer.
#    name : Assigend variable names (used in ObsErr tables).
#    unit : Units of the variables expected in the program.
#    desc : Desriptions of the variables
#    mvar : Multi-variate flags
#
#-------------------------------------------------------------------------------
#////////////////////////////////////////////////////////////////////////////////
#-------------------------------------------------------------------------------
DataTypeTable::

#     mvars
#  kt    name     unit desc       kt = 1 2 3 4 5 6 7

     1 u_SeaLev   m/sec   Sea Level East-West Wind $ 1
     2 v_SeaLev   m/sec   Sea Level South-North Wind $ 1 1
     3 p_SeaLev   hPa     Sea Level Pressure $ 1 1 1
     4 u_UprAir   m/sec   Upper Air East-West Wind $ 0 0 0 1
     5 v_UprAir   m/sec   Upper Air South-North Wind $ 0 0 0 1 1
     6 H_UprAir   m       Upper Air Geopotential Hight $ 0 0 0 1 1 1
     7 q_UprAir   g/kg    Upper Air Water Vapor Mixing Ratio $ 0 0 0 0 0 0 1

:: # End of "DataTypeTable"
#.

#-------------------------------------------------------------------------------
# This file specifies forecast error covariance parameters in the same
# way as the OI-2.5 run of September '96.
#
# Notes:
#
# moisture forecast errors:
# OI analyzes relative humidity while PSAS uses mixing
# ratio. The statistical parameters in this file pertain
# to mixing ratio errors. However the correlations
# (horizontal as well as vertical) are taken the same as
# in the OI run, since our tuning results are not very
# different.
# vertical correlations:
# vertical correlations have been slightly adjusted to ensure
# positive definiteness.
#
# Dick Dee 27Sep96
#-------------------------------------------------------------------------------

FcstErr*hCor_HH::

   DAHP-COSINE GEOS-DAS/OI pseudo multi-level horizontal correlations

# pres  d_m  c1  c2  c3  c4  c5

   1000  6030  .01840  .00389  .98160  .00105 1.20815
      .4 6030  .01840  .00389  .98160  .00105 1.20815

::

FcstErr*hCor_QQ::

   DAHP-COSINE GEOS-DAS/OI pseudo multi-level horizontal correlations
```

```
# pres  d_m  c1  c2  c3  c4  c5

  1000  6030  .0       .0      1.0     .00216  1.9558
    .4  6030  .0       .0      1.0     .00216  1.9558

::

FcstErr*vCor_HH::

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1  .4

  1000   1.
   850   .66  1.
   700   .57  .83  1.
   500   .43  .68  .77  1.
   400   .37  .59  .70  .87  1.
   300   .30  .48  .59  .80  .85  1.
   250   .25  .40  .49  .71  .77  .86  1.
   200   .14  .24  .32  .52  .61  .72  .81  1.
   150   .10  .19  .25  .43  .47  .56  .66  .82  1.
   100   .12  .18  .15  .21  .22  .21  .30  .46  .58  1.
    70   .09  .12  .09  .06  .05  .00  .05  .19  .31  .73  1.
    50   .09  .13  .10  .08  .06  .01  .06  .17  .30  .67  .84  1.
    30   .00  .00  .00  .00  .00  .00  .00  .06  .17  .33  .66  .80  1.
    10   .00  .00  .00  .00  .00  .00  .00  .01  .06  .18  .34  .63  .81  1.
     5   .00  .00  .00  .00  .00  .01  .00 -.01  .01  .04  .19  .35  .63  .81  1.
     2   .00  .00  .00  .00  .00  .00  .00  .00 -.01  .02  .02  .18  .36  .64  .81  1.
     1   .00  .00  .00  .00  .00  .00  .00  .00  .00 -.01  .02  .02  .18  .34  .64  .85  1.
    .4   .00  .00  .00  .00  .00  .00  .00  .00  .00  .00  .00  .02  .04  .18  .33  .68  .82  1.

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1  .4
::

FcstErr*vCor_QQ::

#pres 1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2  1 0.40

1000  1.00
 850  0.27 1.00
 700 -0.01 0.22 1.00
 500  0.04 0.10 0.30 1.00
 400  0.05 0.09 0.16 0.61 1.00
 300  0.04 0.02 0.03 0.22 0.50 1.00
 250  0.00 0.00 0.00 0.00 0.00 0.00 1.00
 200  0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 150  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 100  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  70  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  50  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  30  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  10  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   5  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   2  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   1  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
0.40  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00

#pres 1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2  1 0.40

::

FcstErr*Sigma_Wind::
```

```
#-------------------------------------------------------------------------
#      NASA/GSFC, Data Assimilation Office, Code 910.3, GEOS/DAS      !
#-------------------------------------------------------------------------
#
# !NAME: FcstErr*Sigma_Wind::  Fall '94 NA raob multivariate O-F's (emeasaF)
#
# !DESCRIPTION: Forecast wind error standard deviations: stream function and
# velocity potential components.
#       Tuned to fall 1994 North-American rawinsonde height-wind
# O-F's obtained from the emeasaF run.
# Horizontal correlations modeled by the compactly
# supported spline function.
#
# !REVISION HISTORY:
#  May 13, 1996 - C. Redder and D. Dee - (original table)
#-------------------------------------------------------------------------

NULL

# model* desc

# pres*    sigMs   sigMv

     1000     5.80   13.45
      925     5.71   11.52
      850     5.62    9.59
      700     6.14    7.38
      500     7.61    8.60
      400     8.34   10.92
      300    10.13   12.19
      250    12.69   12.82
      200    15.88   11.67
      150    19.06    9.91
      100    21.12    8.14
       70    22.35    7.10
       50    24.00    6.74
       40    24.98    7.60
       30    25.95    8.46
       20    25.95    8.48
       15    25.95    8.48
       10    25.95    8.48
        7    25.95    8.48
        5    25.95    8.48
        3    25.95    8.48
        2    25.95    8.48
        1    25.95    8.48
      0.7    25.95    8.48
      0.5    25.95    8.48
      0.4    25.95    8.48
      0.2    25.95    8.48
     0.05    25.95    8.48
     0.01    25.95    8.48

::

FcstErr*hCor_SS::

#-------------------------------------------------------------------------
#      NASA/GSFC, Data Assimilation Office, Code 910.3, GEOS/DAS      !
#-------------------------------------------------------------------------
```

```
#
# !NAME: FcstErr*hCor_SS:: Fall '94 NA raob multivariate O-F's (emeasaF)
#
# !DESCRIPTION: Forecast wind error horizontal correlations: stream function
# component.
#              Tuned to fall 1994 North-American rawinsonde height-wind
# O-F's obtained from the emeasaF run.
# Horizontal correlations modeled by the compactly
# supported spline function.
# Constant length scale --> separable correlation model.
#
# !REVISION HISTORY:
#  May 13, 1996 - C. Redder and D. Dee - (original table)
#  Aug 14, 1996 - D. Dee - replaced tuned length scales by a constant
# (tuned 500mb value)
#-----------------------------------------------------------------------

# model* desc

  GASPARI-COHN compactly supported spline function

# pres*    d_m*    L ( km )

  1000     3000     300.0
   925     3000     300.0
   850     3000     300.0
   700     3000     300.0
   500     3000     300.0
   400     3000     300.0
   300     3000     300.0
   250     3000     300.0
   200     3000     300.0
   150     3000     300.0
   100     3000     300.0
    70     3000     300.0
    50     3000     300.0
    40     3000     300.0
    30     3000     300.0
    20     3000     300.0
    15     3000     300.0
    10     3000     300.0
     7     3000     300.0
     5     3000     300.0
     3     3000     300.0
     2     3000     300.0
     1     3000     300.0
   0.7     3000     300.0
   0.5     3000     300.0
   0.4     3000     300.0
   0.2     3000     300.0
  0.05     3000     300.0
  0.01     3000     300.0

::

FcstErr*vCor_SS::

#-----------------------------------------------------------------------
#       NASA/GSFC, Data Assimilation Office, Code 910.3, GEOS/DAS      !
#-----------------------------------------------------------------------
#
```

```
# !NAME: FcstErr*wCor_SS:: Fall '94 NA multivariate O-F's (emeasaF)
#
# !DESCRIPTION: Forecast wind error vertical correlations: stream function
# component.
#      Tuned to fall 1994 North-American rawinsonde height-wind
# O-F's obtained from the emeasaF run.
#
# !REVISION HISTORY:
#  May 13, 1996 - C. Redder and D. Dee - (original table)
#-------------------------------------------------------------------

#pres 1000  925  850  700  500  400  300  250  200  150  100   70   50   40   30   20   15   10    7    5    3    2    1 0.70 0.50 0.40 0.20 0.05 0.01

 1000 1.00
  925 0.00 1.00
  850 0.00 0.00 1.00
  700 0.00 0.00 0.00 1.00
  500 0.00 0.00 0.00 0.00 1.00
  400 0.00 0.00 0.00 0.00 0.00 1.00
  300 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  250 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  200 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  150 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  100 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   70 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   40 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   30 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   20 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   15 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   10 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
    7 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
    5 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
    3 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
    2 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
    1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 0.70 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 0.40 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 0.20 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 0.05 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00

#pres 1000  925  850  700  500  400  300  250  200  150  100   70   50   40   30   20   15   10    7    5    3    2    1 0.70 0.50 0.40 0.20 0.05 0.01

::

FcstErr*hCor_VV::

#-------------------------------------------------------------------
#      NASA/GSFC, Data Assimilation Office, Code 910.3, GEOS/DAS    !
#-------------------------------------------------------------------
#
# !NAME: FcstErr*hCor_VV:: Fall '94 NA multivariate O-F's (emeasaF)
#
# !DESCRIPTION: Forecast wind error horizontal correlations: velocity potential
# component.
#      Tuned to fall 1994 North-American rawinsonde height-wind
# O-F's obtained from the emeasaF run.
# Horizontal correlations modeled by the compactly
```

```
# supported spline function.
# Constant length scale --> separable correlation model.
#
# !REVISION HISTORY:
#  May 13, 1996 - C. Redder and D. Dee - (original table)
#  Aug 14, 1996 - D. Dee - replaced tuned length scales by a constant
# (tuned 500mb value)
#-------------------------------------------------------------------

# model* desc

   GASPARI-COHN multi-level spline horizontal correlations

# pres*   d_x*   L ( km )

    1000    3000    570.0
     925    3000    570.0
     850    3000    570.0
     700    3000    570.0
     500    3000    570.0
     400    3000    570.0
     300    3000    570.0
     250    3000    570.0
     200    3000    570.0
     150    3000    570.0
     100    3000    570.0
      70    3000    570.0
      50    3000    570.0
      40    3000    570.0
      30    3000    570.0
      20    3000    570.0
      15    3000    570.0
      10    3000    570.0
       7    3000    570.0
       5    3000    570.0
       3    3000    570.0
       2    3000    570.0
       1    3000    570.0
     0.7    3000    570.0
     0.5    3000    570.0
     0.4    3000    570.0
     0.2    3000    570.0
    0.05    3000    570.0
    0.01    3000    570.0

::

FcstErr*vCor_VV::

#-------------------------------------------------------------------
#     NASA/GSFC, Data Assimilation Office, Code 910.3, GEOS/DAS     !
#-------------------------------------------------------------------
#
# !NAME: FcstErr*vCor_VV:: Fall '94 NA multivariate O-F's (emeasaF)
#
# !DESCRIPTION: Forecast wind error vertical correlations: velocity potential
# component.      Tuned to fall 1994 North-American rawinsonde height-wind
# O-F's obtained from the emeasaF run.
#
```

```
# !REVISION HISTORY:
#  May 13, 1996 - C. Redder and D. Dee - (original table)
#_____________________________________________________________

#pres 1000  925  850  700  500  400  300  250  200  150  100   70   50   40   30   20   15   10    7    5    3    2    1 0.70 0.50 0.40 0.20 0.05 0.01

1000  1.00
 925  0.00 1.00
 850  0.00 0.00 1.00
 700  0.00 0.00 0.00 1.00
 500  0.00 0.00 0.00 0.00 1.00
 400  0.00 0.00 0.00 0.00 0.00 1.00
 300  0.00 0.00 0.00 0.00 0.00 0.00 1.00
 250  0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 200  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 150  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 100  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  70  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  50  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  40  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  30  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  20  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  15  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  10  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   7  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   5  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   3  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   2  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
   1  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
0.70  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
0.50  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
0.40  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
0.20  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
0.05  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
0.01  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00

#pres 1000  925  850  700  500  400  300  250  200  150  100   70   50   40   30   20   15   10    7    5    3    2    1 0.70 0.50 0.40 0.20 0.05 0.01
::

FcstErr*Aref::
#_____________________________________________________________
# !NAME: FcstErr*Aref:: - a parameter table of mass-wind balance
#
# !DESCRIPTION:
# This model uses the maximum latitude locations to describe the pattern
# of the mass-wind balanced scheme.
#
# !REVISION HISTORY:
#  21Mar96 - J. Guo - the first version.  Numbers are made
# up for testing purposes only.  They are only physically
# possible, but yet to be tuned w.r.t statistics.
#_____________________________________________________________

  GEOS/DAS-OI

# pres A L B

    1000  .4454  .4455  .1
      .4  .4454  .4455  .1
```

```
::

#------------------------------------------------------------------------
# This file specifies observation error covariances in the same way as the
# OI-2.5 run of September '96, with the following exceptions:
#
# moisture observation errors (kx = 7, 9, 10):
# OI analyzes relative humidity while PSAS uses mixing
# ratio. The statistical parameters in this file are
# for mixing ratio errors, obtained by tuning to rawinsonde
# O-F's over Europe (without removing the time mean).
# NESDIS height retrievals over ocean (kx = 39--44):
# in the OI run the standard deviations had been erroneously
# left unchanged from the control run. Here they are set
# equal to the land values.
# vertical correlations:
# vertical correlations have been slightly adjusted to ensure
# positive definiteness.
#
# Dick Dee 27Sep96
#------------------------------------------------------------------------

DataSourceTable::

#------------------------------------------------------------------------
#       NASA/GSFC, Data Assimilation Office, Code 910.3, GEOS/DAS      !
#------------------------------------------------------------------------
#
# !NAME: DataSourceTable:: - a table of instruments with error classes
#
# !REVISION HISTORY:
#
# 04Jan95 - Jing G. - Tabulated version of "kx"
#
#       kx : Original assigned numbers of data sources. int.
#     clas : Original char*8 names for compatibility and
# observation error class.
#     rank : Assigned ranks, integer.
#     desc : A long name or description, char*25.
#------------------------------------------------------------------------

#  kx    clas      rank      desc

    1  SFC_LND1  -10000  Surface Land Obs - 1
    2  SFC_LND2  -10100  Surface Land Obs - 2
    3  SFC_SHP1  -10400  Surface Ship Obs - 1
    4  SFC_SHP2  -10500  Surface Ship Obs - 2
    5  BUOY_ENV  -10200  Environment Buoy
    6  BUOY_DRF  -10300  Drifting Buoy
    7  RAWINSND    -100  Rawinsonde
    8  PILOT_WD    -700  Pilot Balloon Wind
    9  NAVAIDS     -300  Ship Released Rawinsonde
   10  DROPWSND    -500  Dropwinsonde
   11  RADAR_WD    -200  Radar-tracked Rawinsonde
   12  ROCKTSND    -400  Rocketsonde
   13  BALLOON     -800  Balloon
   14  AIR_ASDR   -3000  Aircraft - Air/Sat Relay
   15  AIR_AIDS   -3100  Aircraft - Int. Data Sys
   16  AIR_AIRP   -3200  Aircraft Report
   17  AIR_CODR   -3300  Aircraft Coded Report
   18  AIR_ALPX   -3400  Aircraft - ALPX
```

```
 19  CLDTRKWD  -3500  Cld Trk Wind - Wisc E1
 20  CLDTRKWD  -3600  Cld Trk Wind - Wisc E2
 21  CLDTRKWD  -3700  Cld Trk Wind - Wisc W
 22  CLDTRKWD  -3800  Cld Trk Wind - Wisc Ocean
 23  CLDTRKWD  -3900  Cld Trk Wind - Repr. Jap.
 24  CLDTRKWD  -4000  Cld Trk Wind - NESS East
 25  CLDTRKWD  -4100  Cld Trk Wind - NESS West
 26  CLDTRKWD  -4200  Cld Trk Wind - European
 27  CLDTRKWD  -4300  Cld Trk Wind - Japanese
#28  SEASAT    -10600 SEASAT - Scatterometer
#29  WSAT_55    -600  WSAT 55
#30  WSAT_57    -610  WSAT 57
#31  LIHS      -1100  Limb Infrared Hon.- Strat
#32  VTPR___W -100001 User-defined instrument 1
 33  TRSA_NHL   900  NESDIS NH Land AH type A
 34  TRSA_SHL   901  NESDIS SH Land AH type A
 35  TRSB_NHL   902  NESDIS NH Land AH type B
 36  TRSB_SHL   903  NESDIS SH Land AH type B
 37  TRSC_NHL   904  NESDIS NH Land AH type C
 38  TRSC_SHL   905  NESDIS SH Land AH type C
 39  TRSA_NHW   906  NESDIS NH Ocn  AH type A
 40  TRSA_SHW   907  NESDIS SH Ocn  AH type A
 41  TRSB_NHW   908  NESDIS NH Ocn  AH type B
 42  TRSB_SHW   909  NESDIS SH Ocn  AH type B
 43  TRSC_NHW   910  NESDIS NH Ocn  AH type C
 44  TRSC_SHW   911  NESDIS SH Ocn  AH type C
#45  MO6A_NHL   912  NESDIS NH Land PH type A
#46  MO6A_SHL   913  NESDIS SH Land PH type A
#47  MO6B_NHL   914  NESDIS NH Land PH type B
#48  MO6B_SHL   915  NESDIS SH Land PH type B
#49  MO6C_NHL   916  NESDIS NH Land PH type C
#50  MO6C_SHL   917  NESDIS SH Land PH type C
#51  MO6A_NHW   918  NESDIS NH Ocn  PH type A
#52  MO6A_SHW   919  NESDIS SH Ocn  PH type A
#53  MO6B_NHW   920  NESDIS NH Ocn  PH type B
#54  MO6B_SHW   921  NESDIS SH Ocn  PH type B
#55  MO6C_NHW   922  NESDIS NH Ocn  PH type C
#56  MO6C_SHW   923  NESDIS SH Ocn  PH type C
#57  SPEA_NHW   924  Special Sat NH - Ocn A
#58  SPEA_SHW   925  Special Sat SH - Ocn A
#59  SPEB_NHW   926  Special Sat NH - Ocn B
#60  SPEB_SHW   927  Special Sat SH - Ocn B
#61  SPEC_NHW   928  Special Sat NH - Ocn C
#62  SPEC_SHW   929  Special Sat SH - Ocn C
#63  VASA_NHL  1200  VAS NH Land - type A
#64  VASA_SHL  1201  VAS SH Land - type A
#65  VASB_NHL  1202  VAS NH Land - type B
#66  VASB_SHL  1203  VAS SH Land - type B
#67  VASA_NHW  1204  VAS NH Ocean- type A
#68  VASA_SHW  1205  VAS SH Ocean- type A
#69  VASB_NHW  1206  VAS NH Ocean- type B
#70  VASB_SHW  1206  VAS SH Ocean- type B
#71  GLAA_NHL  1000  NASA-SDUO NH Land  type A
#72  GLAA_SHL  1001  NASA-SDUO SH Land  type A
#73  GLAB_NHL  1002  NASA-SDUO NH Land  type B
#74  GLAB_SHL  1003  NASA-SDUO SH Land  type B
#75  GLAC_NHL  1004  NASA-SDUO NH Land  type C
#76  GLAC_SHL  1005  NASA-SDUO SH Land  type C
#77  GLAD_NHL  1006  NASA-SDUO NH Land  type D
#78  GLAD_SHL  1007  NASA-SDUO SH Land  type D
#79  GLAA_NHW  1008  NASA-SDUO NH Ocean type A
```

```
#  80  GLAA_SHW    1009   NASA-SDUO SH Ocean type A
#  81  GLAB_NHW    1010   NASA-SDUO NH Ocean type B
#  82  GLAB_SHW    1011   NASA-SDUO SH Ocean type B
#  83  GLAC_NHW    1012   NASA-SDUO NH Ocean type C
#  84  GLAC_SHW    1013   NASA-SDUO SH Ocean type C
#  85  GLAD_NHW    1014   NASA-SDUO NH Ocean type D
#  86  GLAD_SHW    1015   NASA-SDUO SH Ocean type D
    87  GEN_1000    -4400  Pseudo-1000mb Heights
#  88  ER2HHS      -3410  ER-2 Aircraft / HHS Data
#  89  ACARS     -100002  ACRAS: Aircraft wind instrument # was User-defined instrument 2
#  90  UNUSED    -100003  User-defined instrument 3
#  91  UARS_HLS    -1300  UARS / HLS 1
#  92  -           -1301  UARS / HLS 2
#  93  -           -1302  UARS -    3
#  94  -           -1303  UARS -    4
#  95  -         -100004  User-defined instrument 4
#  96  -          -10700  SSH/I - 1
#  97  -          -10800  SSH/I - 2
#  98  -            1400  AHSU      NH Land  type A
#  99  -            1401  AHSU      SH Land  type A
# 100  -            1402  AHSU      NH Land  type B
# 101  -            1403  AHSU      SH Land  type B
# 102  -            1404  AHSU      NH Land  type C
# 103  -            1405  AHSU      SH Land  type C
# 104  -            1406  AHSU      NH Land  type D
# 105  -            1407  AHSU      SH Land  type D
# 106  -            1408  AHSU      NH Ocean type A
# 107  -            1409  AHSU      SH Ocean type A
# 108  -            1410  AHSU      NH Ocean type B
# 109  -            1411  AHSU      SH Ocean type B
# 110  -            1412  AHSU      NH Ocean type C
# 111  -            1413  AHSU      SH Ocean type C
# 112  -            1414  AHSU      NH Ocean type D
# 113  -            1415  AHSU      SH Ocean type D

:: # End of DataSourceTable

ObsErr*Levels:     Pressure  1000 850 700 500 400 300 250 200 150 100 70 50 30 10 5 2 1 0.4

# Observation Error data table format
#
# >ObsErr*<kxname>::
# >  <ktname>  <a sig-0 list>
# >  <sec>    <class>
# >  ...
# >::

ObsErr*SFC_LND1::   # kx =  1
   p_SeaLev    1.9
::
ObsErr*SFC_LND2::   # kx =  2
   p_SeaLev    1.9
::
ObsErr*SFC_SHP1::   # kx =  3
   p_SeaLev    1.9
   u_SeaLev    2.8
   v_SeaLev    2.8
::
ObsErr*SFC_SHP2::   # kx =  4
   p_SeaLev    1.9
```

```
   u_SeaLev     2.8
   v_SeaLev     2.8
  ::
  ObsErr*BUOY_ENV::    # kx =   5
     p_SeaLev     1.9
     u_SeaLev     2.8
     v_SeaLev     2.8
  ::
  ObsErr*BUOY_DRF::    # kx =   6
     p_SeaLev     1.9
     u_SeaLev     2.8
     v_SeaLev     2.8
  ::
  ObsErr*RAWINSND::    # kx =   7
     q_UprAir.u   0.76  0.71  0.61  0.24  0.09  0.03
     u_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
     v_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
     H_UprAir.u   4.01  5.10  5.66  7.81  9.32 10.59 12.87 14.08 15.88 18.98 21.31 22.81 24.29 25.69 27.02 28.28 39.50 30.66

  #  vCor_QQ.u       5
     vCor_HH.u       1
  ::
  ObsErr*PILOT_WD::    # kx =   8
     u_UprAir     4.4   5.2   7.2   9.0  11.2  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4
     v_UprAir     4.4   5.2   7.2   9.0  11.2  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4  11.4

     vCor_HH.u       1
  ::
  ObsErr*NAVAIDS::     # kx =   9
     q_UprAir     0.76  0.71  0.61  0.24  0.09  0.03
     u_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
     v_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
     H_UprAir.u   4.01  5.10  5.66  7.81  9.32 10.59 12.87 14.08 15.88 18.98 21.31 22.81

  #  vCor_QQ.u       5
     vCor_HH.u       1
  ::
  ObsErr*DROPWSND::    # kx =  10
     u_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
     v_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
     H_UprAir.u   4.01  5.10  5.66  7.81  9.32 10.59 12.87 14.08 15.88 18.98 21.31 22.81
  ::
  ObsErr*RADAR_WD::    # kx =  11
     q_UprAir     0.76  0.71  0.61  0.24  0.09  0.03

  #  vCor_QQ.u       5
  ::
  ObsErr*ROCKTSND::    # kx =  12
     H_UprAir     3.75  5.2   7.13  7.74  8.74  9.00  9.10 10.30 13.26 17.80 19.52 24.10 24.10 84.96 83.04 89.76 93.60 95.52
     u_UprAir     5.2   6.3   5.6   7.7   7.8  10.7  10.6  11.8  10.7   8.8   7.5   9.0   9.5   1.1   1.2   1.5   1.2   1.9
     v_UprAir     5.2   6.3   5.6   7.7   7.8  10.7  10.6  11.8  10.7   8.8   7.5   9.0   9.5   1.1   1.2   1.5   1.2   1.9
  ::
  ObsErr*BALLOON::     # kx =  13
     u_UprAir     2.2   2.6   3.2   4.0   6.0   8.0   9.0  10.0   9.5   9.0   8.0   7.0   8.6   9.7   9.7   9.7   9.7   9.7
     v_UprAir     2.2   2.6   3.2   4.0   6.0   8.0   9.0  10.0   9.5   9.0   8.0   7.0   8.6   9.7   9.7   9.7   9.7   9.7
  ::
  ObsErr*AIR_ASDR::    # kx =  14
     u_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
     v_UprAir.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
  ::
  ObsErr*AIR_AIDS::    # kx =  15
```

```
  u_Uprair.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
  v_Uprair.u   1.7   2.0   2.2   2.4   2.9   3.2   3.4   3.3   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7   2.7
::
ObsErr*AIR_AIRP::    # kx = 16
    u_Uprair     3.1   3.6   3.6     4.5   5.6   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7
    v_Uprair     3.1   3.6   3.6     4.5   5.6   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7
::
ObsErr*AIR_CODR::    # kx = 17
    u_Uprair     3.1   3.6   3.6     4.5   5.6   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7
    v_Uprair     3.1   3.6   3.6     4.5   5.6   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7   5.7
::
ObsErr*AIR_ALPX::    # kx = 18
::
ObsErr*CLDTRWD::     # kx = 19--27
  u_Uprair.u   4.64  4.62  4.13  3.60  5.80  6.00  6.50  6.50  6.50  7.00  7.00  7.00  7.00  7.00
  v_Uprair.u   4.64  4.62  4.13  3.60  5.80  6.00  6.50  6.50  6.50  7.00  7.00  7.00  7.00  7.00
::
ObsErr*TRSA_WHL::    # kx = 33
  H_Uprair.u   4.3   8.6   10.7  12.0  12.8  14.1  15.0  16.1  19.7  20.2  23.6  23.8  25.5  27.2  29.1  31.0  32.9  34.8
  H_Uprair.c   5.3   10.5  13.1  14.6  15.7  17.3  18.4  19.7  24.1  24.7  28.9  29.2  31.2  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u    3
  vCor_HH.c    3
  hCor_HH.c    1
::
ObsErr*TRSA_SHL::    # kx = 34
  H_Uprair.u   4.3   8.6   10.7  12.0  12.8  14.1  15.0  16.1  19.7  20.2  23.6  23.8  25.5  27.2  29.1  31.0  32.9  34.8
  H_Uprair.c   5.3   10.5  13.1  14.6  15.7  17.3  18.4  19.7  24.1  24.7  28.9  29.2  31.2  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u    3
  vCor_HH.c    3
  hCor_HH.c    1
::
ObsErr*TRSB_WHL::    # kx = 35
  H_Uprair.u   4.3   8.6   10.7  12.0  12.8  14.1  15.0  16.1  19.7  20.2  23.6  23.8  25.5  27.2  29.1  31.0  32.9  34.8
  H_Uprair.c   5.3   10.5  13.1  14.6  15.7  17.3  18.4  19.7  24.1  24.7  28.9  29.2  31.2  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u    3
  vCor_HH.c    3
  hCor_HH.c    1
::
ObsErr*TRSB_SHL::    # kx = 36
  H_Uprair.u   4.3   8.6   10.7  12.0  12.8  14.1  15.0  16.1  19.7  20.2  23.6  23.8  25.5  27.2  29.1  31.0  32.9  34.8
  H_Uprair.c   5.3   10.5  13.1  14.6  15.7  17.3  18.4  19.7  24.1  24.7  28.9  29.2  31.2  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u    3
  vCor_HH.c    3
  hCor_HH.c    1
::
ObsErr*TRSC_WHL::    # kx = 37
  H_Uprair.u   5.6   9.8   13.6  17.7  20.0  21.6  22.8  23.3  24.1  23.1  22.9  23.8  25.6  27.2  29.1  31.0  32.9  34.8
  H_Uprair.c   6.9   12.0  16.7  21.7  24.5  26.4  28.0  28.5  29.5  28.2  28.1  29.2  31.4  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u    4
  vCor_HH.c    4
  hCor_HH.c    1
::
ObsErr*TRSC_SHL::    # kx = 38
  H_Uprair.u   5.6   9.8   13.6  17.7  20.0  21.6  22.8  23.3  24.1  23.1  22.9  23.8  25.6  27.2  29.1  31.0  32.9  34.8
  H_Uprair.c   6.9   12.0  16.7  21.7  24.5  26.4  28.0  28.5  29.5  28.2  28.1  29.2  31.4  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u    4
  vCor_HH.c    4
  hCor_HH.c    1
::
ObsErr*TRSA_WHW::    # kx = 39
  H_Uprair.u   4.3   8.6   10.7  12.0  12.8  14.1  15.0  16.1  19.7  20.2  23.6  23.8  25.5  27.2  29.1  31.0  32.9  34.8
```

```
  H_UprAir.c    5.3 10.5  13.1  14.6  15.7  17.3  18.4  19.7  24.1  24.7  28.9  29.2  31.2  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u     3
  vCor_HH.c     3
  hCor_HH.c     1
::
ObsErr*TRSA_SHW::    # kx =  40
  H_UprAir.u    4.3  8.6  10.7  12.0  12.8  14.1  15.0  16.1  19.7  20.2  23.6  23.8  25.5  27.2  29.1  31.0  32.9  34.8
  H_UprAir.c    5.3 10.5  13.1  14.6  15.7  17.3  18.4  19.7  24.1  24.7  28.9  29.2  31.2  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u     3
  vCor_HH.c     3
  hCor_HH.c     1
::
ObsErr*TRSB_NHW::    # kx =  41
  H_UprAir.u    4.3  8.6  10.7  12.0  12.8  14.1  15.0  16.1  19.7  20.2  23.6  23.8  25.5  27.2  29.1  31.0  32.9  34.8
  H_UprAir.c    5.3 10.5  13.1  14.6  15.7  17.3  18.4  19.7  24.1  24.7  28.9  29.2  31.2  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u     3
  vCor_HH.c     3
  hCor_HH.c     1
::
ObsErr*TRSB_SHW::    # kx =  42
  H_UprAir.u    4.3  8.6  10.7  12.0  12.8  14.1  15.0  16.1  19.7  20.2  23.6  23.8  25.5  27.2  29.1  31.0  32.9  34.8
  H_UprAir.c    5.3 10.5  13.1  14.6  15.7  17.3  18.4  19.7  24.1  24.7  28.9  29.2  31.2  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u     3
  vCor_HH.c     3
  hCor_HH.c     1
::
ObsErr*TRSC_NHW::    # kx =  43
  H_UprAir.u    5.6  9.8  13.6  17.7  20.0  21.6  22.8  23.3  24.1  23.1  22.9  23.8  25.6  27.2  29.1  31.0  32.9  34.8
  H_UprAir.c    6.9 12.0  16.7  21.7  24.5  26.4  28.0  28.5  29.5  28.2  28.1  29.2  31.4  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u     4
  vCor_HH.c     4
  hCor_HH.c     1
::
ObsErr*TRSC_SHW::    # kx =  44
  H_UprAir.u    5.6  9.8  13.6  17.7  20.0  21.6  22.8  23.3  24.1  23.1  22.9  23.8  25.6  27.2  29.1  31.0  32.9  34.8
  H_UprAir.c    6.9 12.0  16.7  21.7  24.5  26.4  28.0  28.5  29.5  28.2  28.1  29.2  31.4  33.3  35.6  38.0  40.3  42.6
  vCor_HH.u     4
  vCor_HH.c     4
  hCor_HH.c     1
::
ObsErr*GEM_1000::    # kx =  87
  H_UprAir     10.0  18.5  27.2  36.2  40.5  50.5  55.2  59.8  77.7  95.7 136.0 159.0
::

ObsErr*hCor_HH-1::

  GAUSSIAN exp(-(d/L)^2)

# pres  d_m  L

  1000 4800 600
     .4 4800   600
::

ObsErr*vCor_HH-1::

# 05oct95 - Jing G. - This version of vertical correlation function
# is remodeled based a GEOS/DAS-v1.2 18 level table.  The original
# table (see 10Jan95 comment below) is not P.D..
#
# 10Jan95 - Jing G. -
```

```
#
#    Title from GEOS_DAS data table file "vcoroe.data" reads,
#
# > RAWINSONDE VCORS FROM HL, LH (LIKE VCOROE2 W/ EIGEN VALS >=.1)
#

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1   .4

 1000   1.
  850   .48  1.
  700   .36  .64  1.
  500   .15  .24  .70  1.
  400   .09  .14  .55  .79  1.
  300   .05  .13  .40  .63  .80  1.
  250   .03  .14  .42  .60  .74  .90  1.
  200   .08  .22  .44  .52  .60  .73  .79  1.
  150   .07  .21  .41  .51  .58  .68  .72  .87  1.
  100   .03  .11  .33  .47  .53  .64  .67  .77  .88  1.
   70   .02  .06  .28  .42  .50  .60  .63  .72  .80  .88  1.
   50   .00  .04  .24  .40  .48  .59  .61  .68  .74  .82  .89  1.
   30  -.01  .02  .04  .05  .07  .16  .21  .44  .56  .65  .72  .78  1.
   10  -.01 -.01  .01  .04  .05  .09  .10  .26  .43  .57  .66  .73  .88  1.
    5   .00 -.03 -.01  .02  .02  .04  .04  .11  .26  .44  .58  .67  .80  .88  1.
    2   .00 -.02 -.02 -.01  .00  .00  .00  .05  .12  .27  .45  .60  .72  .79  .87  1.
    1   .00  .00 -.02 -.02 -.03 -.04 -.05  .00  .06  .13  .28  .49  .65  .71  .77  .86  1.
   .4   .00  .02  .00 -.03 -.04 -.10 -.11 -.07  .00  .06  .15  .33  .56  .64  .69  .74  .84  1.

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1   .4
::
ObsErr*wCor_HH-3:::

# 05oct95 - Jing G. - This version of vertical correlation function
# is remodeled based a GEOS/DAS-v1.2 18 level table.  The original
# table (see 10Jan95 comment below) is not P.D..
#
# 10Jan95 - Jing G. -
#
#    Title from GEOS_DAS data table file "vcoroe.data" reads,
#
# > VERTICAL CORR OF TIROS A,B, AND VTPR HEIGHT ERROR (EIGEN
# > VALS >=.1)

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1   .4

 1000   1.
  850   .00  1.
  700   .00  .82  1.
  500   .00  .52  .72  1.
  400   .00  .36  .56  .85  1.
  300   .00  .19  .37  .73  .84  1.
  250   .00  .16  .33  .64  .76  .87  1.
  200   .00  .15  .29  .54  .63  .74  .81  1.
  150   .00  .22  .33  .47  .52  .57  .66  .79  1.
  100   .00  .16  .24  .35  .38  .42  .53  .66  .80  1.
   70   .00  .07  .16  .23  .27  .31  .41  .55  .67  .80  1.
   50   .00  .12  .20  .26  .28  .31  .40  .52  .65  .76  .84  1.
   30   .00  .00  .01  .00  .00  .02  .10  .35  .51  .64  .75  .80  1.
   10   .00  .00  .01  .00 -.01 -.01  .02  .10  .36  .51  .64  .74  .84  1.
    5   .00  .00  .00  .00 -.01 -.02 -.01  .03  .10  .36  .51  .63  .75  .83  1.
    2   .00 -.01  .00  .00  .00 -.01 -.02  .00  .03  .10  .36  .50  .64  .75  .83  1.
```

```
     1  .00  .00  .00  .00  .00  .00 -.02 -.01 -.01  .03  .10  .36  .53  .64  .75  .83  1.
    .4  .00  .00 -.01  .00  .01  .01 -.02 -.02 -.03 -.03  .02  .13  .41  .52  .63  .74  .82  1.

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1   .4
::

ObsErr*vCor_HH-4::

# 05oct95 - Jing G. - This version of vertical correlation function
# is remodeled based a GEOS/DAS-v1.2 18 level table.  The original
# table (see 10Jan95 comment below) is not P.D.
#
# 10Jan95 - Jing G. -
#
#    Title from GEOS_DAS data table file "vcoroe.data" reads,
#
# > VERTICAL CORR OF TIROS C HEIGHT ERROR (EIGEN VALS >=.1)

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1   .4

    1000  1.
     850  .00  1.
     700  .00  .83  1.
     500  .00  .67  .81  1.
     400  .00  .51  .67  .85  1.
     300  .00  .29  .47  .72  .84  1.
     250  .00  .20  .39  .64  .77  .88  1.
     200  .00  .17  .33  .54  .65  .76  .81  1.
     150  .00  .20  .31  .46  .54  .63  .70  .80  1.
     100  .00  .19  .29  .39  .46  .51  .58  .69  .80  1.
      70  .00  .12  .22  .30  .35  .38  .46  .58  .70  .81  1.
      50  .00  .15  .22  .29  .33  .35  .42  .52  .65  .76  .84  1.
      30  .00  .00  .01  .00  .01  .04  .11  .35  .50  .63  .74  .79  1.
      10  .00  .01  .01  .00  .00 -.01  .03  .11  .36  .50  .63  .74  .84  1.
       5  .00  .00  .01  .00  .00 -.02 -.01  .03  .11  .37  .50  .63  .75  .83  1.
       2  .00  .00  .00  .00  .00 -.01 -.02  .00  .04  .11  .36  .50  .64  .75  .83  1.
       1  .00  .00 -.01  .00  .00  .00 -.02 -.01  .00  .04  .11  .37  .52  .64  .75  .83  1.
      .4  .00  .00 -.01  .00  .00  .01 -.02 -.02 -.02  .02  .13  .41  .51  .63  .73  .81  1.

# pres  1000  850  700  500  400  300  250  200  150  100   70   50   30   10    5    2    1   .4
::

ObsErr*vCor_QQ-5::

#-----------------------------------------------------------------
#      NASA/GSFC, Data Assimilation Office, Code 910.3, GEOS/DAS    !
#-----------------------------------------------------------------
#
# !NAME: ObsErr*vCor_QQ-5::  Fall '94 EU mixing ratio O-F's (emeasaF)
#
# !DESCRIPTION: Raob mixing ratio error vertical correlations.
#       Tuned to fall 1994 O-F's over Europe obtained from
# the emeasaF run (bias not removed).
#
# !REVISION HISTORY:
# May 14, 1996 - C. Redder and D. Dee - (original table)
# May 21, 1996 - D. Dee - addition of 10mb vertical correlations
# Sep 27, 1996 - D. Dee - very slight changes due to retuning
#-----------------------------------------------------------------

#pres 1000  850  700  500  400  300  250  200  150  100   70   50   30   10
```

```
1000  1.00
 850  0.24 1.00
 700  0.10 0.16 1.00
 500  0.00 0.10 0.10 1.00
 400  0.00 0.00 0.00 0.27 1.00
 300  0.00 0.00 0.00 0.10 0.34 1.00
 250  0.00 0.00 0.00 0.00 0.00 0.00 1.00
 200  0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 150  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
 100  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  70  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  50  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  30  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
  10  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
#pres 1000  850  700  500  400  300  250  200  150  100   70   50   30   10
::
```

Appendix D: Complete Covariance Data Life Cycle Diagrams

Complete covariance data life cycle diagrams for the operators Γ^h, Σ^h, Σ^ψ, Σ^χ, C^h, C^ψ, C^χ, Σ_u^o, Σ_c^o, C_u^o, and C_c^o are presented in Figures 53-58 on pages 182 through 187.

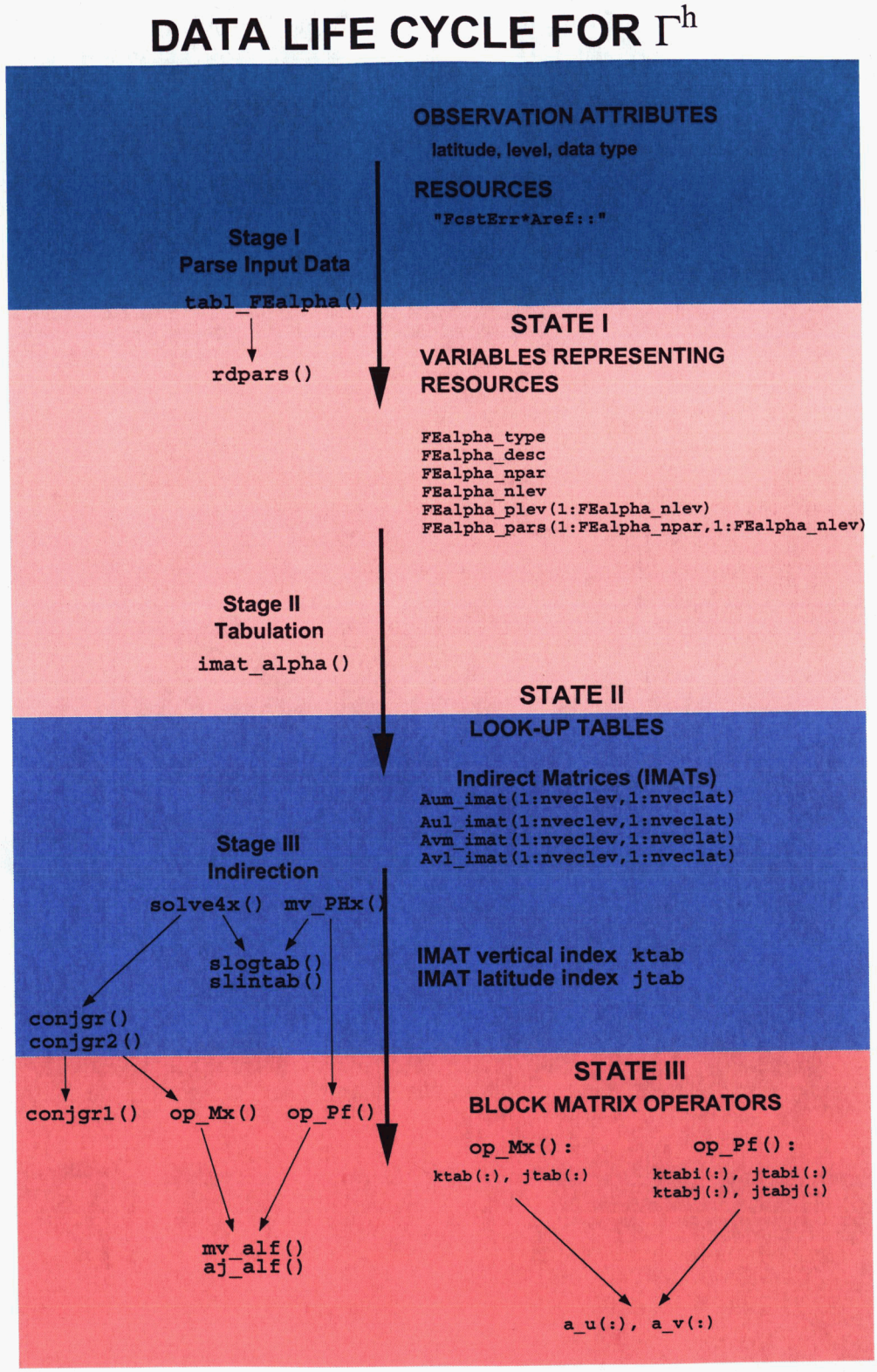


Figure 53: Data life cycle for the operator Γ^h.

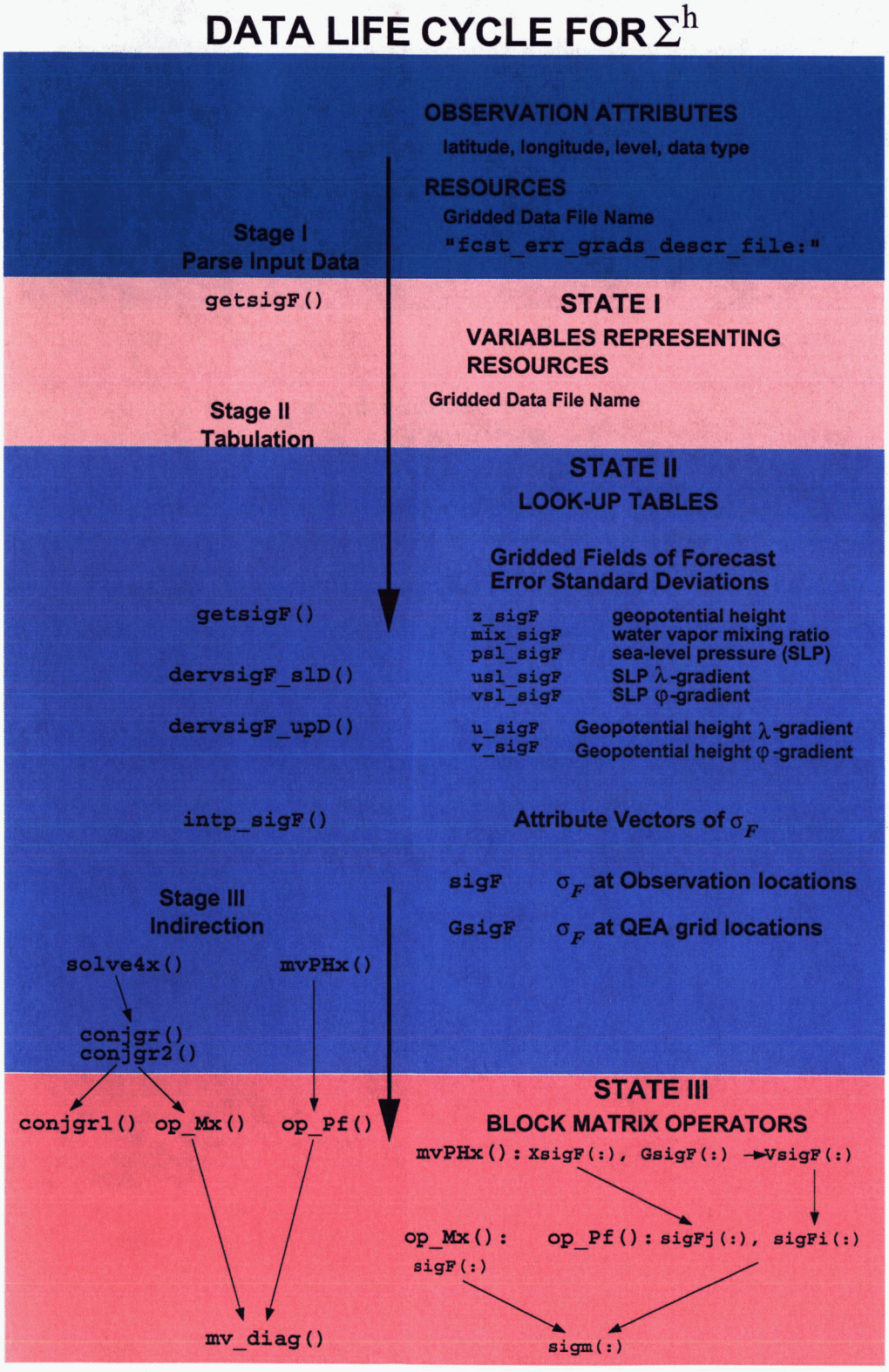


Figure 54: Data life cycle for the operator Σ^h.

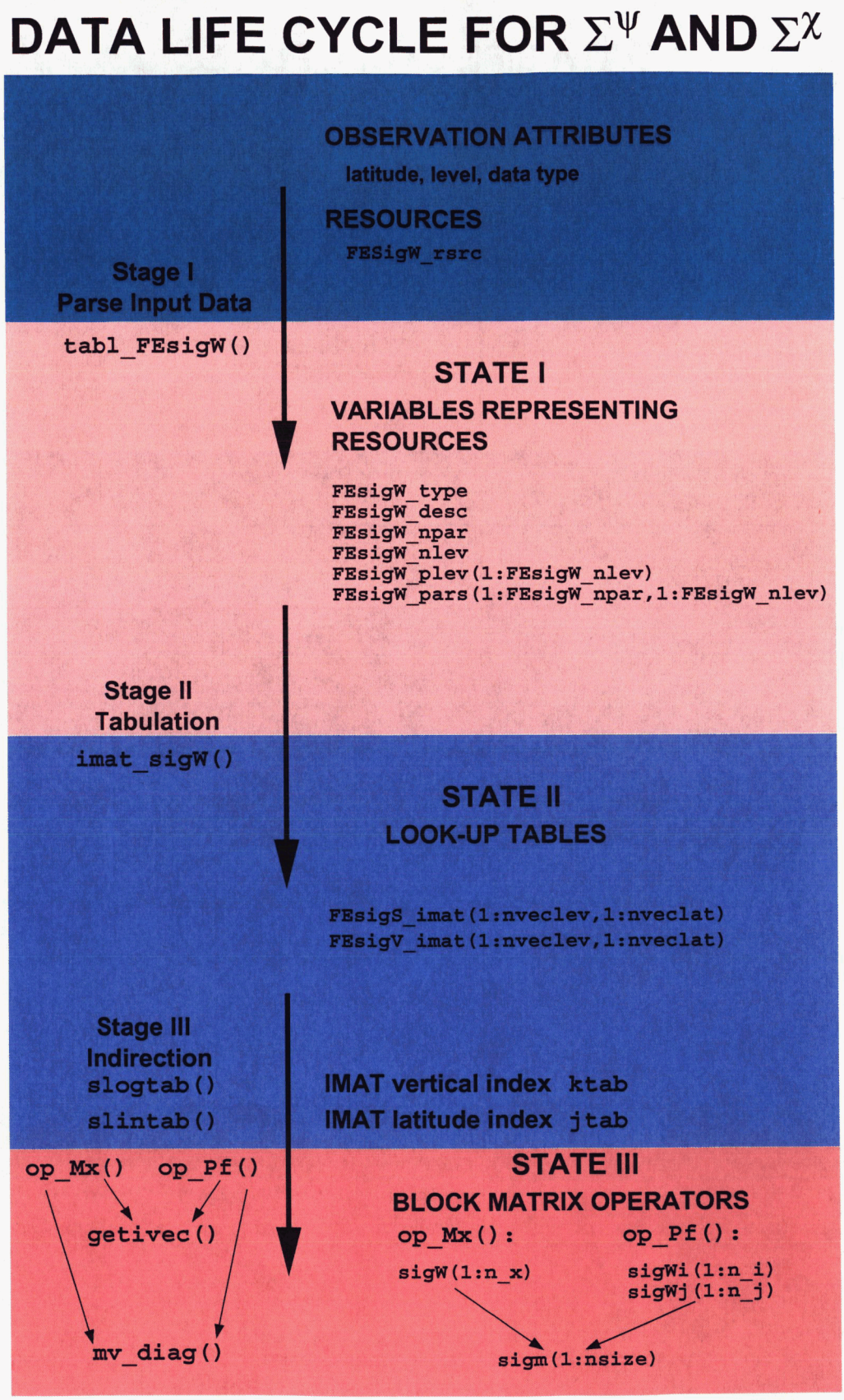


Figure 55: Data life cycle for the operators Σ^{ψ} and Σ^{χ}.

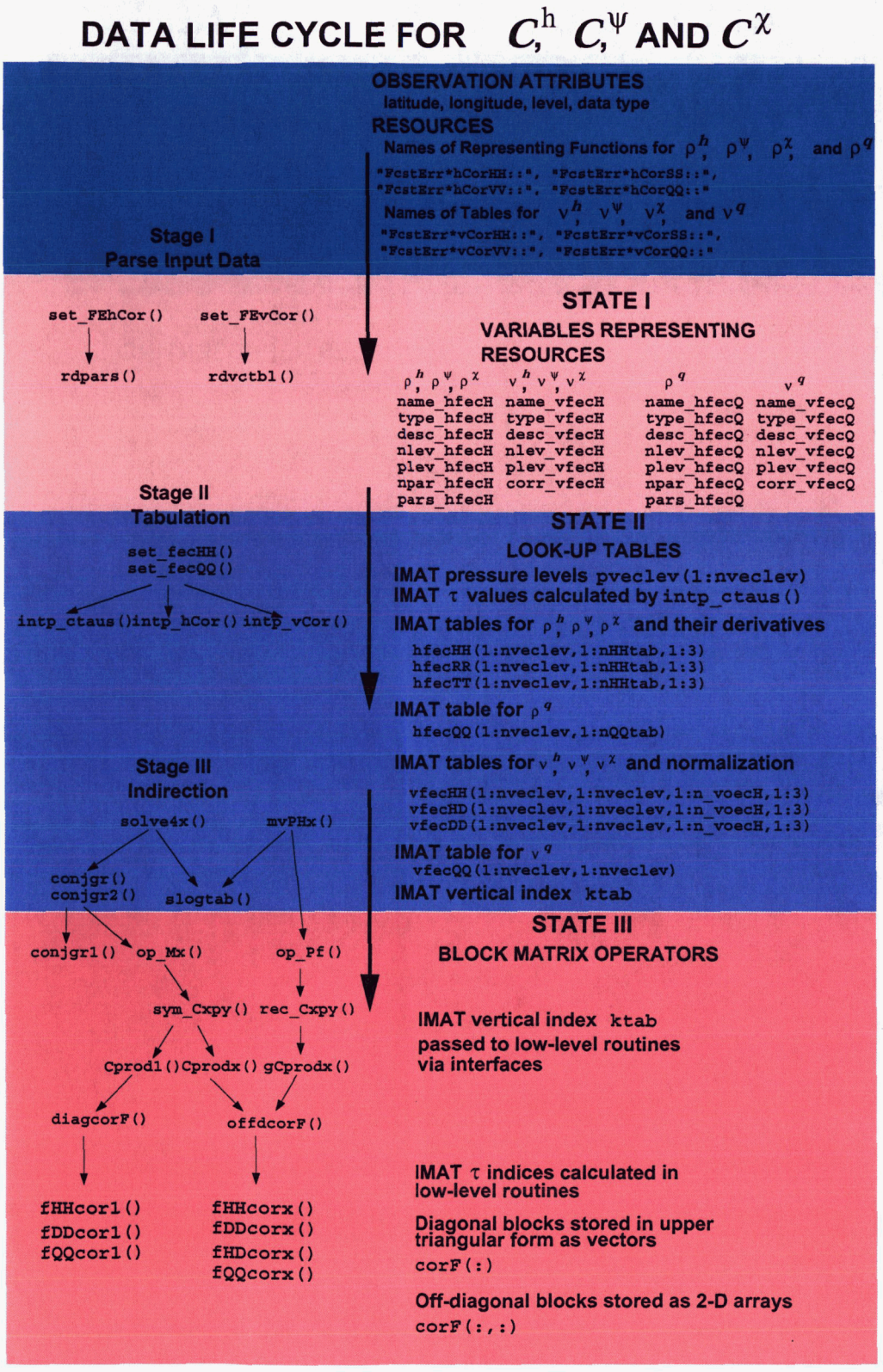


Figure 56: Data life cycle for the operators C^h, C^ψ, and C^χ.

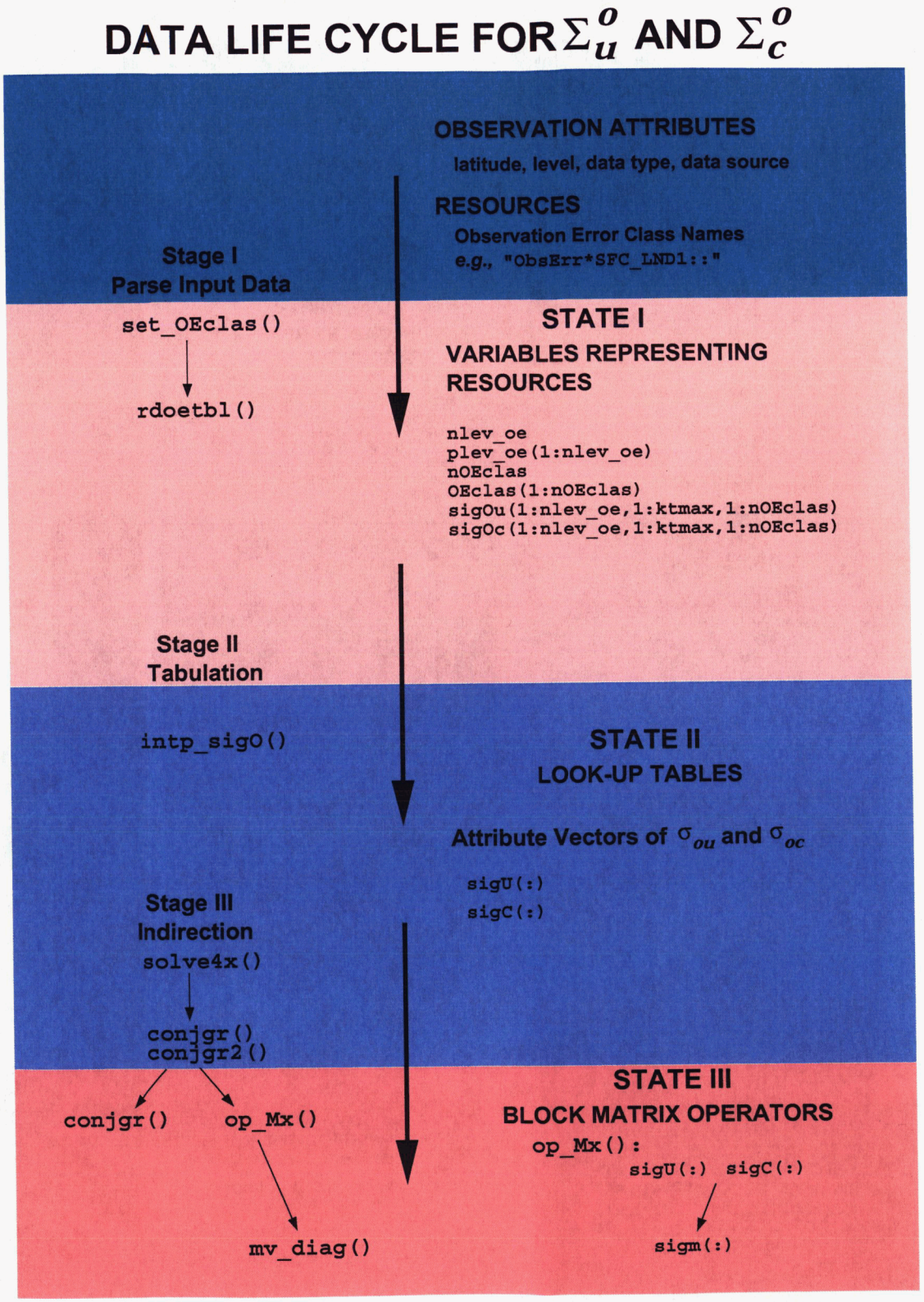


Figure 57: Data life cycle for the operators Σ_u^o and Σ_c^o.

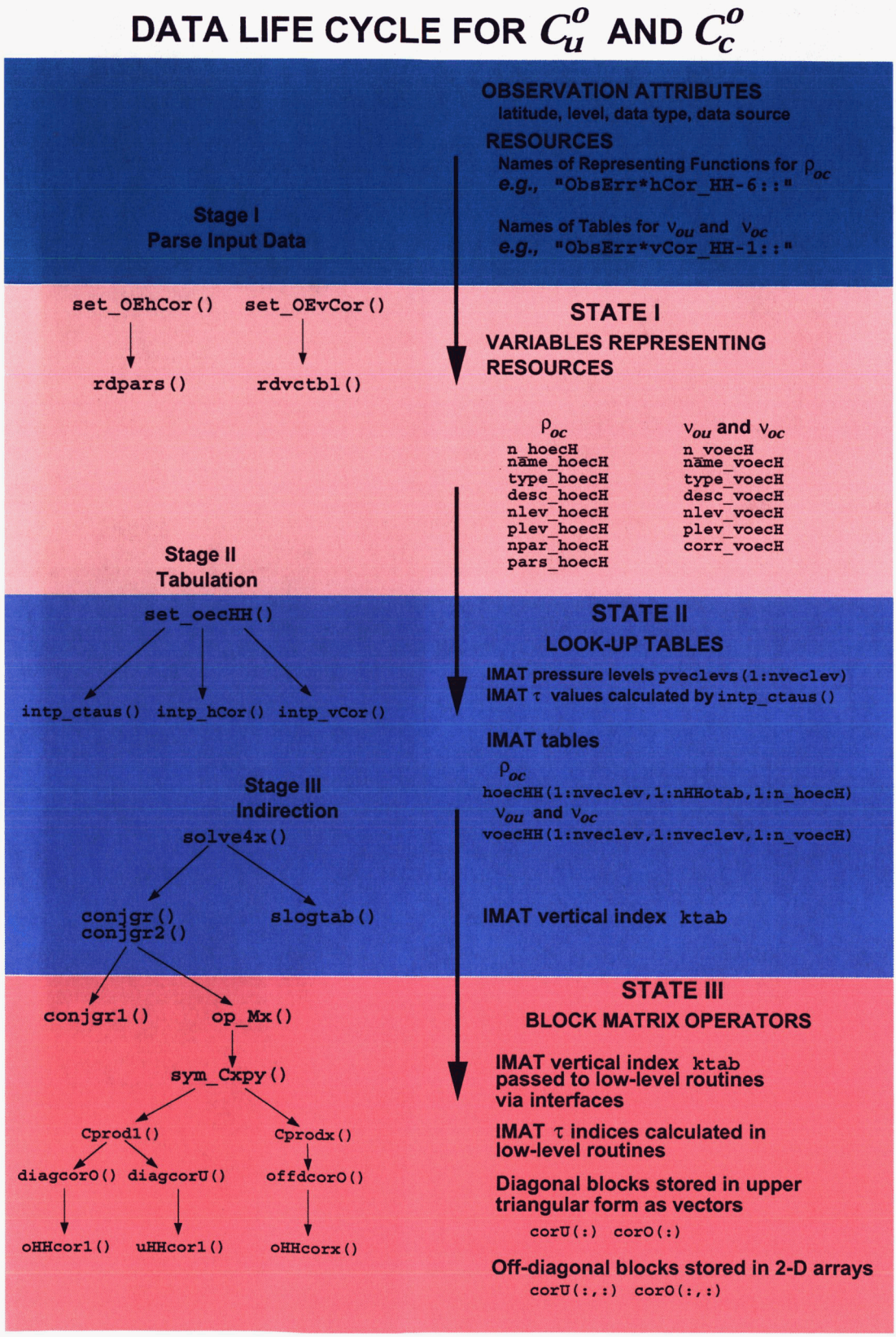


Figure 58: Data life cycle for the operators C_u^o and C_c^o.

Acknowledgments

We would like to thank Stephen Cohn, David Lamich, Richard Rood, and Meta Sienkiewicz for their considerable help during the course of this work. We thank also the attendees of the PSAS workshop on October 26, 1998 for their helpful comments. The research and development documented in this article were supported by the NASA EOS Interdisciplinary Science Program, the NASA Research and Applications Program, and the NASA High Performance Computing and Communications (HPCC) Earth and Space Sciences (ESS) program. Computer resources and funding were provided by the EOS Program through the Data Assimilation Office and by the HPCC program.

References

[Cohn et al., 1998] Cohn, S. E., da Silva, A., Guo, J., Sienkiewicz, M., and Lamich, D. (1998). Assessing the effects of data selection with the DAO Physical-space Statistical Analysis System. *Mon. Wea. Rev.*, 126:2913–2926.

[da Silva and Guo, 1996] da Silva, A. and Guo, J. (1996). Documentation of the Physical-space Statistical Analysis System (PSAS) Part I: The Conjugate Gradient Solver, Version PSAS-1.00. Technical Report DAO Office Note No. 96-02 **http://dao.gsfc.nasa.gov/subpages/office-notes.html**, NASA/Goddard Space Flight Center, Greenbelt, Maryland.

[da Silva et al., 1999] da Silva, A., Tippett, M., and Guo, J. (1999). The PSAS Users' Manual. Technical Report To be published as DAO Office Note 99-XX, NASA/Goddard Space Flight Center, Greenbelt, Maryland.

[Gaspari and Cohn, 1999] Gaspari, G. and Cohn, S. E. (1999). Construction of Correlation Functions in Two and Three Dimensions. *Quart. J. Roy. Met. Soc.*, 125:723–757.

[Gaspari et al., 1998] Gaspari, G., Cohn, S. E., Dee, D. P., Guo, J., and da Silva, A. M. (1998). Construction of the PSAS Multi-level Forecast Error Covariance Models. Technical Report DAO Office Note 98-06 **http://dao.gsfc.nasa.gov/subpages/office-notes.html**, NASA/Goddard Space Flight Center, Greenbelt, Maryland.

[Golub and van Loan, 1989] Golub, G. H. and van Loan, C. F. (1989). *Matrix Computations*. The John Hopkins University Press, Baltimore, second edition.

[Guo et al., 1998] Guo, J., Larson, J. W., Gaspari, G., da Silva, A., and Lyster, P. M. (1998). Documentation of the Physical-space Statistical Analysis System (PSAS) Part II: The Factored-Operator Formulation of Error Covariances. Technical Report DAO Office Note 98-04 **http://dao.gsfc.nasa.gov/subpages/office-notes.html**, NASA/Goddard Space Flight Center, Greenbelt, Maryland.

[Lyster, 1998] Lyster, P. M. (1998). The Computational Complexity of Atmospheric Data Assimilation. *Submitted to Int. J. Appl. Sci. Comp.* Available on-line from **http://dao.gsfc.nasa.gov/DAO_people/lys/complexity**.

[Pfaendtner, 1996] Pfaendtner, J. W. (1996). Notes on the Icosahedral Domain Decompostion in PSAS. Technical Report DAO Office Note 96-04 **http://dao.gsfc.nasa.gov/subpages/office-notes.html**, NASA/Goddard Space Flight Center, Greenbelt, Maryland.

[Staff, 1996] Staff, D. A. O. (1996). Algorithm Theoretical Basis Document, Version 1.01. Technical report, NASA/Goddard Space Flight Center, Greenbelt, Maryland **http://dao.gsfc.nasa.gov/subpages/atbd.html**.