DAO Office Note 1998-05

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.

<table>
<thead>
<tr>
<th>Version Number</th>
<th>Version Date</th>
<th>Pages Affected/ Extent of Changes</th>
<th>Approval Authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1β</td>
<td>October 26, 1998</td>
<td>First draft (before review) from PSAS Workshop</td>
<td></td>
</tr>
<tr>
<td>Version 1</td>
<td>December 6, 1999</td>
<td>Initial release</td>
<td>ON Editor</td>
</tr>
</tbody>
</table>
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


Visit also the Data Assimilation Office's Home Page at

http://dao.gsfc.nasa.gov/
DAO Office Note 1998-05

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—knameO() 33
  5.3 Observation Data Source Tables—xxnameO() 33
  5.4 Determination of Observation Error Classes and State I data for \( \sigma_u \) and \( \sigma_o \)—set_0Eclas() 35
  5.5 Initialization of State I Observation Error Vertical Correlation Tables 40
    5.5.1 State I Observation Error Vertical Correlations—set_0EvCor() 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 $\sigma^v$ and $\sigma^x$—tabl_FEsigW() .................................................. 51
5.8 State I Data for the Geostrophic Balance Parameters $\alpha_{ul}$, $\alpha_{um}$, and $\alpha_{el}$—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—dupelimit() ................................................. 58
6.4 Superobbing—proxel() ........................................................................................................... 62
6.5 Separation of Eliminated and Retained Observations—tcfront() ........................................... 64

7 Stage II: Processing of Covariance Data ......................................................................... 68
7.1 Interpolation of Observation Error Standard Deviations from State I Tables—intp.sig0() .................................................. 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 $\alpha_{um}$, $\alpha_{ul}$, $\alpha_{um}$, and $\alpha_{el}$—imat.alpha() .................................................. 83
7.6 IMAT Tables for $\sigma^v$ and $\sigma^x$—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 $\Sigma^h$—dervsigF.s1D() .................................................. 87
7.7.3 Gridded Fields of the Elements of $\Sigma^h$—dervsigF.upD() .................................................. 89
7.7.4 Interpolation of Elements of $\Sigma^h$—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—con1gr() ....................................................................... 95
8.3 Software Implementation of $(HP^fH^T + R)x$—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 $P^fH^T x$—op_Pf() .................................................. 114
10 Stage III: Software Implementation the "Lower-Level" Operators in PSAS120

10.1 Mass-coupled Height/Wind Forecast Error Covariances $\Gamma^H$ ............... 120
10.2 Mass-decoupled Height/Wind Error Covariances $\Gamma^V$ and $\Gamma^X$ ........... 122
10.3 Forecast/Observation Error Standard Deviation Operator $\Sigma$ ................. 125
10.4 The Symmetric Error Correlation Operator – sym.Cxy() .......................... 125
   10.4.1 The Diagonal Block Operator Cprodi() ...................................... 131
   10.4.2 The Off-diagonal Block Operator Cprod() ................................... 134
10.5 The Rectangular Error Correlation Operator $C_{rec}$ ............................ 135
   10.5.1 The Block Operator gcprod() .................................................. 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 ktnameO(). .............................................................................. 34
12. Calling tree for kxxnameO(). ............................................................................ 35
13. Calling tree for setOEclas(). ........................................................................... 39
14. Observation error standard deviation data structures. .................................. 39
15. Calling tree for setOEevCor(). ......................................................................... 42
16. Data structures containing State I tables for $u_{0}$ and $v_{0}$. ....................... 43
17. Calling tree for setOEhEvCor(). ....................................................................... 44
18. Data structures containing State I Data for $p_{0}$. ........................................... 45
19. Calling tree for setFEevCor(). ......................................................................... 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 tabI_FEsigW(). ....................................................................... 52
26. State I data for the streamfunction and velocity potential forecast error standard deviations $\sigma^v$ and $\sigma^x$. ........................................... 53
27. Calling tree for tabl_FEalpha(). ..................................................................... 54
28. State I data for the geostrophic balance parameters $\alpha_{um}$, $\alpha_{ul}$, $\alpha_{vm}$, and $\alpha_{vl}$. ................................................................. 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_sigQ(). .............................................................................. 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_s1D(). ................................................................. 88
41. Calling tree for dervsigF_upD(). ................................................................. 90
List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data type index $\mathbf{kt}$ values</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Nonzero elements of the multivariate operator $\Gamma^h$</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>Nonzero elements of the multivariate operator $\Gamma^\psi$</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>Nonzero elements of the multivariate operator $\Gamma^x$</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Elements of the diagonal matrix $\Sigma^h = q \hat{\sigma}^h$</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>Elements of the diagonal matrix $\Sigma^\psi$</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>Elements of the diagonal matrix $\Sigma^x$</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Nonzero elements of the operator $\mathbf{C}^A$</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>Nonzero elements of the operator $\mathbf{C}^\psi$</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Nonzero elements of the operator $\mathbf{C}^x$</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>Data passed into getAIall() via its interface.</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>Data box attributes $\mathbf{boxes}$ returned by setbox().</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>Data passed into restrict() via its interface.</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>Data passed into sort() via its interface.</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>Data passed into dupelim() via its interface.</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>Black list data structures from proxi1.h</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>Data passed into proxi1() via its interface.</td>
<td>65</td>
</tr>
<tr>
<td>9</td>
<td>Data passed into tofront() via its interface.</td>
<td>67</td>
</tr>
<tr>
<td>10</td>
<td>Data passed into intp.sig0() via its interface.</td>
<td>70</td>
</tr>
<tr>
<td>11</td>
<td>Data passed into merg.plevs() via its interface.</td>
<td>72</td>
</tr>
<tr>
<td>12</td>
<td>Data passed into merg.lats() via its interface.</td>
<td>74</td>
</tr>
<tr>
<td>13</td>
<td>Forecast Height/Wind Error Horizontal Correlation IMATs</td>
<td>77</td>
</tr>
<tr>
<td>14</td>
<td>Forecast Height/Wind Error Vertical Correlation IMATs</td>
<td>78</td>
</tr>
<tr>
<td>15</td>
<td>Forecast Height/Wind Error Correlation Normalization Factors</td>
<td>79</td>
</tr>
<tr>
<td>16</td>
<td>Objects from the module FEalpha_imat</td>
<td>84</td>
</tr>
<tr>
<td>17</td>
<td>IMATs for $\sigma^\psi$ and $\sigma^x$</td>
<td>85</td>
</tr>
<tr>
<td>18</td>
<td>Data passed into getsig() via its interface.</td>
<td>87</td>
</tr>
<tr>
<td>19</td>
<td>Data passed into dervsigF_swD() via its interface.</td>
<td>88</td>
</tr>
<tr>
<td>20</td>
<td>Data passed into dervsigF_upD() via its interface.</td>
<td>90</td>
</tr>
<tr>
<td>21</td>
<td>Data passed into intp.sigF() via its interface.</td>
<td>93</td>
</tr>
<tr>
<td>22</td>
<td>Summary of arguments to solve4x().</td>
<td>97</td>
</tr>
<tr>
<td>23</td>
<td>Summary of arguments to op.Mx().</td>
<td>102</td>
</tr>
<tr>
<td>24</td>
<td>Arguments to the routine getAinc().</td>
<td>109</td>
</tr>
<tr>
<td>25</td>
<td>Arguments to the routine mvPHx().</td>
<td>113</td>
</tr>
<tr>
<td>26</td>
<td>Summary of arguments to op.Pf().</td>
<td>119</td>
</tr>
<tr>
<td>27</td>
<td>Summary of arguments to mv.Allf().</td>
<td>122</td>
</tr>
<tr>
<td>28</td>
<td>Summary of arguments to mv.diag().</td>
<td>126</td>
</tr>
<tr>
<td>29</td>
<td>Summary of arguments to sym_Cpy().</td>
<td>132</td>
</tr>
<tr>
<td>30</td>
<td>Error Correlation Calculation Calls from Cprodi().</td>
<td>132</td>
</tr>
<tr>
<td>31</td>
<td>Summary of arguments to Cprodi().</td>
<td>133</td>
</tr>
<tr>
<td>32</td>
<td>Error Correlation Calculation Calls from Cprodx().</td>
<td>135</td>
</tr>
<tr>
<td>33</td>
<td>Summary of arguments to Cprodx().</td>
<td>136</td>
</tr>
<tr>
<td>34</td>
<td>Summary of arguments to rec_Cpy(). The 'columns' arguments n.kr.j, q[i,j,y] are not shown for brevity.</td>
<td>140</td>
</tr>
<tr>
<td>35</td>
<td>Summary of arguments to gCprodx().</td>
<td>142</td>
</tr>
<tr>
<td>36</td>
<td>Data passed into getAไพuv() via its interface.</td>
<td>144</td>
</tr>
</tbody>
</table>
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:


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 $w^f$ and observations $w^o$ to produce an optimal estimate of the true state of the atmosphere. The PSAS performs this analysis using the following equations:

The innovation equation:

$$(HP^fHT + R)x = w^o - Hw^f;$$

and

The analysis equation:

$$w^a - w^f = P^fHTx,$$

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, $w^o$ is a $p$-vector of observations, $w^f$ is an $n$-vector of the gridded forecast, and $w^a$ 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 $x$ 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 $HP^fHT + R$ and solution of Eqn. (1) for $x$ by a pre-conditioned conjugate gradient method.

2. Construction of the matrix $P^fHT$ and calculation of the AI $(w^a - w^f)$ from $x$ 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 getAlall(). 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 getAlpuv(), the upper-air height-wind analysis getAlzuv(), and the water vapor mixing ratio analysis getAlmix().
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 $HP^T H^T$ and $R$.

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

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

**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.
GENERAL CALLING TREE FOR GEOS-2 PSAS

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 \( \varphi \) and longitude \( \lambda \), 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 \( r \), \( \lambda \), and \( \varphi \) are \( \hat{e}_r \), \( \hat{e}_\lambda \), and \( \hat{e}_\varphi \), respectively. The longitudinal unit vector \( \hat{e}_\lambda \) points to the East, while the meridional unit vector \( \hat{e}_\varphi \) points to geographic North, as illustrated in Figure 2. The set of unit vectors \( (\hat{e}_r, \hat{e}_\lambda, \hat{e}_\varphi) \) form a right-handed triple, since

\[
\hat{e}_r \times \hat{e}_\lambda = \hat{e}_\varphi, \\
\hat{e}_\lambda \times \hat{e}_\varphi = \hat{e}_r, \\
\hat{e}_\varphi \times \hat{e}_r = \hat{e}_\lambda.
\]

The horizontal error correlations between observation or forecast values at locations \( (\varphi_i, \lambda_i) \) and \( (\varphi_j, \lambda_j) \) are computed using the polarity index \( r_{ij} \) which is defined as

\[
\tau_{ij} = e_r^i \cdot e_r^j.
\]  

(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 \textit{observed minus forecast} (OMF) vector \( \mathbf{w}^o - H \mathbf{w}^f \in \mathbb{R}^p \). This input OMF vector and the following data form the set of \textit{input observational attributes}:

- Variable type index \( k_t \) (INTEGER), one of the set \{1,2,...,\( k_{t\max} \)\}, where the value of \( k_{t\max} \) is defined in the include file \( k_{t\max}.h \).
- Instrument index \( k_x \) (INTEGER), one of the set \{1,2,...,\( k_{x\max} \)\}, where the value of \( k_{x\max} \) is defined in the include file \( k_{x\max}.h \).
- Region index \( k_r \) (INTEGER, assigned at run-time), one of the set \{1,2,...,\( k_{r\max} \)\}, where the value of \( k_{r\max} \) is defined in the include file \( k_{r\max}.h \).
- Latitude \( \varphi \) (REAL), in degrees (-90,90).
- Longitude \( \lambda \) (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 \textit{output observational attributes}:

- Forecast error standard deviations at observation locations \( \sigma_f \) (REAL).
Figure 2: The spherical coordinate system used in the PSAS.

- Correlated observation error standard deviations $\sigma_{oc}$.
- 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 $\mathbf{ks}$ (INTEGER), corresponding to fixed $(\varphi, \lambda)$.
- Correlated and uncorrelated observation error standard deviations $\sigma_{oc}$ and $\sigma_{ou}$, respectively (REAL).
- Forecast error standard deviations at observation locations $\sigma_f$.
- Cartesian components of the unit vectors $\hat{e}_r$, $\hat{e}_i$, and $\hat{e}_m$, evaluated at observation locations (REAL).
Observation elimination flag \( k_1 \) (LOGICAL).

Vertical and latitudinal indices to covariance data look-up tables used to compute \( H P f H^T \), \( R \), and \( P f H^T \) (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 \( \text{maxreg} \) non-overlapping regions. An icosahedral decomposition (Figure 4) with \( \text{maxreg} = 80 \) is used \cite{Pfaendtner, 1996}. The regions are labeled by the index \( kr \) from 1 to \( \text{maxreg} \), where \( \text{maxreg} \) is defined in the include file \text{maxreg.h}. Each of the above attribute \( p \)-vectors is sorted into \( \text{maxreg} \) regional segments numbered \( kr \), according to which region number the observation is located in. Given \( kr \) from the set of \( \{1,2, \ldots, \text{maxreg}\} \), the integer \( \text{iregbeg}(kr) \) points to the beginning of region \( kr \), and \( \text{ireglen}(kr) \) is the number of observations in region \( kr \). Thus, \( \text{iregbeg}(kr) + \text{ireglen}(kr) - 1 \) 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 \( \{1,2, \ldots, \text{maxreg}\} \) of regions, and \( kt \) from the set of \( \{1,2, \ldots, \text{ktmax}\} \) of data types, the integer \( \text{itypbeg}(kt, kr) \) points to the beginning of the data type segment region \( kt \) in region \( kr \), and \( \text{ityplen}(kt, kr) \) is the number of points in this segment. The index \( \text{itypbeg}(kt, kr) \) is rarely used, since for \( kt = 1 \), \( \text{itypbeg}(1, kr) = \text{iregbeg}(kr) \), and for \( kt \) one of \( \{2, \ldots, \text{ktmax}\} \),

\[
\text{itypbeg}(kt, kr) = \text{iregbeg}(kr) + \sum_{i=1}^{kt-1} \text{ityplen}(i, kr).
\]

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 \text{gridxx.h}, and are currently:

- \( j \text{dimx} \): the number of latitude bands. Currently \( j \text{dimx} = 91 \), and \( \Delta \varphi = 2^\circ \). For \( j = 1, \ldots, j \text{dimx} \) the latitude \( \varphi_j \) for band \( j \) is

\[
\varphi_j = -90. + (j - 1) \Delta \varphi.
\]

- \( i \text{dimx} \): the number of longitude bands. Currently \( i \text{dimx} = 144 \), and \( \Delta \lambda = 2.5^\circ \). For \( i = 1, \ldots, i \text{dimx} \) the longitude \( \lambda_i \) is

\[
\lambda_i = -180. + (i - 1) \Delta \lambda.
\]

- \( k \text{dimx} \): the maximum number of vertical levels. Currently \( k \text{dimx} = 29 \), 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 \( j_{\text{dimx}} \), with the latitude values \( \varphi_1, \ldots, \varphi_{j_{\text{dimx}}} \) defined in Eqn. (4).
- The QEA and analysis gridpoints coincide for latitudes \( |\varphi| \leq \text{eaytresh} \), where \( \text{eaytresh} \)
is a threshold, input at runtime from the PSAS resource file. These latitude bands
have index $j$ in the set $\{j_{\text{south}}, \ldots, j_{\text{north}}\}$.

- QEA and analysis gridpoints in the latitude bands at the North and South poles
  coincide. These latitude bands have index $j = 1$ and $j = \text{idimx}$ for the South and
  North poles, respectively.

- For latitude bands with index $j$ one of $\{2, \ldots, j_{\text{south}} - 1\}$ or $\{j_{\text{north}} + 1, \ldots, \text{idimx} - 1\}$,
n  the number of longitude points $\text{nlon}$ is

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

  \hspace{1cm} (6)

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

  $$\lambda_i = -180.0 + (i - 1) \times \text{dlon}.$$  

The QEA grid parameters and are stored in the common blocks $\text{gridprm}$ and $\text{gridarr}$, defined in the include file $\text{qea.h}$:

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

The horizontal QEA grid is initialized using the following procedure:

1. The latitude threshold $\text{eaytresh}$ is read from the PSAS resource file by the routine
   $\text{gridxx()}$. 

Figure 4: Icosahedral decomposition used in the PSAS, with region numbering.
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 `ealon`. The array `ealon` 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 `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 `maxreg^2ktmax^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 `(kr_i,kt_i,kri,jtj)`, where `i = 1,...,maxreg*ktmax` and `j = 1,...,maxreg*ktmax` are the row and column indices, respectively. The dimensions of the `(i,j)th` `kr/kt` block are:

- `ityplen(kt_i,kr_i)` rows
- `ityplen(kt_j,kr_j)` columns
- `ityplen(kt_i,kr_i)*ityplen(kt_j,kr_j)` elements.
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 $C^h$, $C^\psi$, and $C^x$ defined below approximately 40% full.

### 3.3 Operator Formulation of PSAS

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

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

$$HP^H = \Gamma^h\Sigma^h\Sigma^h\Gamma^h + \Gamma^\psi\Sigma^\psi\Sigma^\psi\Gamma^\psi + \Gamma^x\Sigma^x\Sigma^x\Gamma^x,$$

(7)

where $p_{kt}$ 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 \( \psi \). 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 \( \chi \).

The univariate water mixing ratio forecast error covariance matrix is written

\[
HPHT = \Sigma^qC^qC^qT.
\]  

(8)

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

\[
R = R_c + R_u = \Sigma^2C^2\Sigma^2_c + \Sigma^2_uC^2_u\Sigma^2_u.
\]  

(9)

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

Each of the factors in the terms in Eqn. (7) are \( p_{kt} \times p_{kt} \) matrices, and the ordering of their rows and columns is defined by the \textit{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 \( \Gamma^h \) is \textit{kr/kt} block-diagonal, and has three nonzero bands in each of the diagonal \textit{kr/kt} block. Multiplication of a \( p_{kt} \)-vector by \( \Gamma^h \) requires \( O(p_{kt}) \) operations.

2. \( \Gamma^\psi \) and \( \Gamma^x \) in Eqn. (7) are \textit{kr/kt} block-diagonal, and each have two nonzero bands. Multiplication of a \( p_{kt} \)-vector by \( \Gamma^\psi \) or \( \Gamma^x \) requires \( O(p_{kt}) \) operations.
Table 1: Data type index $kt$ values

<table>
<thead>
<tr>
<th>Variable</th>
<th>$kt$-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sea-level zonal wind $u_{sl}$</td>
<td>1</td>
</tr>
<tr>
<td>Sea-level meridional wind $v_{sl}$</td>
<td>2</td>
</tr>
<tr>
<td>Sea-level pressure $p_{sl}$</td>
<td>3</td>
</tr>
<tr>
<td>Upper-air zonal wind $u$</td>
<td>4</td>
</tr>
<tr>
<td>Upper-air meridional wind $v$</td>
<td>5</td>
</tr>
<tr>
<td>Upper-air geopotential height $h$</td>
<td>6</td>
</tr>
<tr>
<td>Upper-air water vapor mixing ratio $q$</td>
<td>7</td>
</tr>
</tbody>
</table>

3. $\Sigma^h, \Sigma^\psi$, and $\Sigma^x$ are diagonal. Multiplication of a $p_{kt}$-vector by $\Sigma^h, \Sigma^\psi$, or $\Sigma^x$ requires $O(p_{kt}^2)$ operations.

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

5. The matrices $\Sigma^c_o$ and $\Sigma^c_u$ are diagonal.

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

7. The matrix $C^c_u$ is a dense matrix. Multiplication of a $p_{kt}$-vector by $C^c_u$ requires $O(p_{kt}^2)$ operations.

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

$$ P^f HT = \Gamma^h \Sigma^h C^h R \Sigma^h R^T + \Gamma^\psi \Sigma^\psi C^\psi R \Sigma^\psi R^T + \Gamma^x \Sigma^x C^x R \Sigma^x R^T, $$

where $n_{kt}$ 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 $n_{kt} \times n_{kt}$ and $p_{kt} \times p_{kt}$ respectively. The matrices $C^h, C^\psi$, and $C^x$ have dimension $n_{kt} \times p_{kt}$. The above implications numbered 1 and 2 apply to the $p_{kt} \times p_{kt}$ matrices, multiplication of an $n_{kt}$-vector by each of the $n_{kt} \times n_{kt}$ matrices in Eqn. (10) requires $O(n_{kt}^2)$ operations, and the multiplication of an $p_{kt}$-vector by each of the $n_{kt} \times p_{kt}$ matrices in Eqn. (10) requires $O(n_{kt} p_{kt})$ operations.

The definitions of the $kt$ values used in these tables are given in Table 1. The non-zero elements of $\Gamma^h, \Gamma^\psi, \Gamma^x, \Sigma^h, \Sigma^\psi, \Sigma^x, C^h, C^\psi$, and $C^x$ 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 $\Gamma^h$

<table>
<thead>
<tr>
<th>Row kt-value $k_t$</th>
<th>Column kt-value $k_t$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$\alpha_{um}(\varphi, p_{sl})$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>$\alpha_{sl}(\varphi, p_{sl})$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$\alpha_{um}(\varphi, p_{sl})$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$\alpha_{sl}(\varphi, p_{sl})$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$\alpha_{um}(\varphi, p)$</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>$\alpha_{ll}(\varphi, p)$</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>$\alpha_{um}(\varphi, p)$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>$\alpha_{ll}(\varphi, p)$</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: Nonzero elements of the multivariate operator $\Gamma^\psi$

<table>
<thead>
<tr>
<th>Row kt-value $k_t$</th>
<th>Column kt-value $k_t$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$-1$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$-1$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4: Nonzero elements of the multivariate operator $\Gamma^x$

<table>
<thead>
<tr>
<th>Row kt-value $k_t$</th>
<th>Column kt-value $k_t$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 5: Elements of the diagonal matrix $\Sigma^h$ $(\sigma_{Pst} = g\bar{\rho}\sigma^h)$

<table>
<thead>
<tr>
<th>kt-value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\sigma_{Pst}\sqrt{\rho^h(1)}$</td>
</tr>
<tr>
<td>2</td>
<td>$\sigma_{Pst}\sqrt{\rho^h(1)}$</td>
</tr>
<tr>
<td>3</td>
<td>$\sigma_{Pst}$</td>
</tr>
<tr>
<td>4</td>
<td>$g\sigma^h\sqrt{\rho^h(1)}$</td>
</tr>
<tr>
<td>5</td>
<td>$g\sigma^h\sqrt{\rho^h(1)}$</td>
</tr>
<tr>
<td>6</td>
<td>$2\Omega^h$</td>
</tr>
<tr>
<td>7</td>
<td>$\sigma^q$</td>
</tr>
</tbody>
</table>

Table 6: Elements of the diagonal matrix $\Sigma^\psi$

<table>
<thead>
<tr>
<th>kt-value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\sigma^\psi$</td>
</tr>
<tr>
<td>2</td>
<td>$\sigma^\psi$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$\sigma^\psi$</td>
</tr>
<tr>
<td>5</td>
<td>$\sigma^\psi$</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7: Elements of the diagonal matrix $\Sigma^x$

<table>
<thead>
<tr>
<th>kt-value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\sigma^x$</td>
</tr>
<tr>
<td>2</td>
<td>$\sigma^x$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$\sigma^x$</td>
</tr>
<tr>
<td>5</td>
<td>$\sigma^x$</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 8: Nonzero elements of the operator $C^h$

<table>
<thead>
<tr>
<th>Row $kt$-value $kt_i$</th>
<th>Column $kt$-value $kt_j$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$c^{h}_{mm}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>$c^{h}_{ml}$</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>$c^{h}_{mh}$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$c^{h}_{mm}$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$c^{h}_{l}$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>$c^{h}_{h}$</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$c^{h}_{hm}$</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>$c^{h}_{h}$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$c^{h}_{hh}$</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$c^{h}_{mm}$</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>$c^{h}_{ml}$</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>$c^{h}_{mh}$</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>$c^{h}_{lm}$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>$c^{h}_{l}$</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>$c^{h}_{h}$</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>$c^{h}_{hm}$</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>$c^{h}_{hl}$</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>$c^{h}_{hh}$</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>$c^{h}$</td>
</tr>
</tbody>
</table>

Table 9: Nonzero elements of the operator $C^\psi$

<table>
<thead>
<tr>
<th>Row $kt$-value $kt_i$</th>
<th>Column $kt$-value $kt_j$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$c^{\psi}_{mm}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>$c^{\psi}_{ml}$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$c^{\psi}_{mm}$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$c^{\psi}_{l}$</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$c^{\psi}_{mm}$</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>$c^{\psi}_{ml}$</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>$c^{\psi}_{lm}$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>$c^{\psi}_{l}$</td>
</tr>
</tbody>
</table>
Table 10: Nonzero elements of the operator $C^x$

<table>
<thead>
<tr>
<th>Row $k_t$-value $k_t_i$</th>
<th>Column $k_t$-value $k_t_j$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$c_{mm}^x$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>$c_{ml}^x$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$c_{lm}^x$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$c_{ll}^x$</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$c_{mm}^x$</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>$c_{ml}^x$</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>$c_{lm}^x$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>$c_{ll}^x$</td>
</tr>
</tbody>
</table>

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 $\sigma_{oc}$ and $\sigma_{ou}$.
- Observation error horizontal correlations $\rho_{oc}$.
- Horizontally correlated and horizontally uncorrelated observation error vertical correlations $\nu_{oc}$ and $\nu_{ou}$.
- Forecast error standard deviations $\sigma^h$, $\sigma^{vl}$, $\sigma^q$, $\sigma^\psi$, and $\sigma^x$.
- Elements of the coupled mass-balance operator for forecast height/wind errors $\alpha_{um}$, $\alpha_{vm}$, $\alpha_{ul}$, and $\alpha_{vl}$.
DAO Office Note 1998-05

- 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 \( \alpha_{um} \), \( \alpha_{um} \), \( \alpha_{ul} \), and \( \alpha_{vl} \).
  - level-dependent tables of parameters for functions used to model \( \sigma^y \) and \( \sigma^x \).

- **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 \( \Sigma^h \) in Eqn. (7) and \( \Sigma^h_L \) and \( \Sigma^h_R \) in Eqn. (10).
  - attribute vectors of \( \sigma_{uz} \) and \( \sigma_{uc} \), which form the diagonals of \( \Sigma^o_u \) and \( \Sigma^o_c \), 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 \( \tau \) defined in Eqn. (3) above.
    * Observation error vertical correlation coefficients.
    * Observation error horizontal correlation functions.
    * Forecast error standard deviations \( \sigma^y \) and \( \sigma^x \).
    * The geostrophic balance parameters \( \alpha_{um} \), \( \alpha_{um} \), \( \alpha_{ul} \), and \( \alpha_{vl} \).

- **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 1 Processing** is the parsing of input data from the PSAS resource file. This processing stage is discussed in Section 5.
- **Stage 2 Processing** is the refinement and tabulation of State I data, resulting in State II data. This processing stage is discussed in Section 7.
- **Stage 3 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. 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:

\[
\text{latitude\_treshold\_for\_equal\_area\_grid: 45}
\]

The routine gridx0() uses the following block of code to read the value of eaytresh:

\[
\begin{align*}
\text{call LABLIN ( 'latitude\_treshold\_for\_equal\_area\_grid:' )} \\
eaytresh = \text{abs(FLTGET (90.))}
\end{align*}
\]

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].
4 Top Level Control Flow — getAIall()

4.1 Analysis Interfaces to the PSAS

The analysis increments (AI), i.e., the quantities denoted \( w^a - w^f \) 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 \( x \).
- **getAinc()**: Uses Eqn. (2) to \( x \) 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.
DAO Office Note 1998-03

Version 1 Dated 12/06/1999

- **MXvec1at (INTEGER)**: maximum size of `veclats`, the array of IMAT latitudes. Defined in the module `config`
- **nvec1at (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 in 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 Al's 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:
**DAO Office Note 1998-05**

- **sigU_list(npieces):** Horizontally uncorrelated observation error standard deviations $\sigma_{ou}$ (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 $n = 1, \ldots, npieces$ 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 `sigU` and `sigO` in the routine `intp_sigO()`:

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

The tables `sigU` and `sigO` are defined in the module `OEclas.tbl`. The interpolated `sigU` is returned in `sigU_list(1:npieces)` and the interpolated `sigO` is returned in `sigO_list(1:npieces)`.
Table 2: Data passed into `getAIall()` via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>npieces</td>
<td>INTEGER</td>
<td>INOUT</td>
<td>Number of Observations</td>
</tr>
<tr>
<td>lat_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Latitude</td>
</tr>
<tr>
<td>lon_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Longitude</td>
</tr>
<tr>
<td>pres_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Pressure (hPa)</td>
</tr>
<tr>
<td>time_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>$\Delta t$ (min) from Analysis Time</td>
</tr>
<tr>
<td>kx_list</td>
<td>INTEGER(npieces)</td>
<td>INOUT</td>
<td>Data Source Index $kx$</td>
</tr>
<tr>
<td>kt_list</td>
<td>INTEGER(npieces)</td>
<td>INOUT</td>
<td>Data Type Index $kt$</td>
</tr>
<tr>
<td>dels_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>$w^o - Hw^I$</td>
</tr>
<tr>
<td>sigF_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Forecast Error Variances $\sigma_f$ at obs. locations</td>
</tr>
<tr>
<td>sig0_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Observation Error Variances $\sigma_o$</td>
</tr>
<tr>
<td>im</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Longitudinal Grid Points</td>
</tr>
<tr>
<td>jnp</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Latitudinal Grid Points</td>
</tr>
<tr>
<td>mlev</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Grid Vertical Levels</td>
</tr>
<tr>
<td>preslev</td>
<td>REAL(mlev)</td>
<td>INOUT</td>
<td>Analysis Pressure Levels</td>
</tr>
<tr>
<td>ps1sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast Sea-Level Pressure Error Variances</td>
</tr>
<tr>
<td>usl.sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast $u_{sl}$ Error Variances</td>
</tr>
<tr>
<td>vsl.sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast $v_{sl}$ Error Variances</td>
</tr>
<tr>
<td>z.sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast Geopotential Height Error Variances</td>
</tr>
<tr>
<td>usigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast $u$ Error Variances</td>
</tr>
<tr>
<td>vsigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast $v$ Error Variances</td>
</tr>
<tr>
<td>mix.sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast Mixing Ratio Error Variances</td>
</tr>
<tr>
<td>ps1.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Sea-Level Pressure Analysis Increments</td>
</tr>
<tr>
<td>usl.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$u_{sl}$ Analysis Increments</td>
</tr>
<tr>
<td>vsl.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$v_{sl}$ Analysis Increments</td>
</tr>
<tr>
<td>z.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Geopotential Height Analysis Increments</td>
</tr>
<tr>
<td>u.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$u$ Analysis Increments</td>
</tr>
<tr>
<td>v.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$v$ Analysis Increments</td>
</tr>
<tr>
<td>mix.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Mixing Ratio Analysis Increments</td>
</tr>
<tr>
<td>ps1sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Sea-Level Pressure Analysis Error Variances</td>
</tr>
<tr>
<td>usl.sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$u_{sl}$ Analysis Error Variances</td>
</tr>
<tr>
<td>vsl.sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$v_{sl}$ Analysis Error Variances</td>
</tr>
<tr>
<td>z.sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Geopotential Height Analysis Error Variances</td>
</tr>
<tr>
<td>usigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$u$ Analysis Error Variances</td>
</tr>
<tr>
<td>vsigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$v$ Analysis Error Variances</td>
</tr>
<tr>
<td>mix.sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Mixing Ratio Analysis Error Variances</td>
</tr>
</tbody>
</table>
Initialization and Stage I Processing of Covariance Data

Processing of Observation Attributes

Stage II Processing of Covariance Data

Solution of Innovation Eqn.

Calculation of AI

Calculation of Wind Error Standard Deviations

Stage III Processing of Covariance Data

Diagnostic Output

Figure 9: Top-level control flow below analysis interfaces `getA1all()`. 

25
4.3.2 Processing of Observational Data

A call to the routine \texttt{restrict()} selects the observations used to calculate the analysis increments.

\begin{verbatim}
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 )
\end{verbatim}

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

The value of \texttt{npieces} is set to \texttt{nnobs} for subsequent processing.

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

\begin{verbatim}
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 )
\end{verbatim}

Further discussion of the routine \texttt{sort()} is given in Section 6.2.

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

\begin{verbatim}
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, ityplen )
\end{verbatim}

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

\begin{verbatim}
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, & sigU_list, sigU_list, sigF_list, time_list, &
\end{verbatim}
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 `preslev(1:mlev)`, and the observation pressure level values `pres_list(1:npieces)`. This union is found through a pair of calls to the routine `mergeplevs()`:  

```fortran
  call mergplevs(mlev,preslev,1,pres4slp,MXveclev,nveclev,pveclev)
  call mergplevs(npieces,pres_list,nveclev,pveclev,MXveclev,nveclev,pveclev)
```

The first call to `mergeplevs()` 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 `mergeplevs()`, 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 `mergeplevs()`. Further discussion of `mergeplevs()` 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 `merglats()`.  

```fortran
  call merglats(jnp,npieces,lat_list,MXveclat,nveclat,veclats)
```

The number of imat latitudes `nveclat` ≤ `MXveclat`. Further discussion of the routine `merglats()` 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 IMAT's `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 `v fecQQ`, 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 $a_{um}$, $a_{ul}$, $a_{um}$, and $a_{ul}$. These data are contained in the IMATs $A_{um}.imat$, $A_{ul}.imat$, $Av_{m}.imat$, and $Av_{l}.imat$, and computed by a call to `imat.alpha()` (see Section 7.5).

IMAT tables of forecast error standard deviations $\sigma^v$ and $\sigma^x$. These data are contained in the IMATs $FE_{sigS}.imat$ and $FE_{sigV}.imat$, 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):

```fortran
CALL getsigF(im,jnp,mlev,pres_lev,psl_sигF,z_sигF,mix_sигF)
```

The routine `getsigF()` returns:

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

For the sea-level analysis, gridded fields of the quantity $\sigma^p = \sqrt{\rho(1)/2\Omega \bar{\rho}a}$ are required in order to calculate the forecast error covariance operator. For the sea-level analysis, this quantity is calculated by calling `dervsigF_s1D()` (see Section 7.7.2):

```fortran
CALL dervsigF_s1D(im,jnp,psl_sигF,usl_sигF,vsl_sигF)
```

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

```fortran
CALL dervsigF_upD(im,jnp,mlev,pres_lev,z_sигF,u_sигF,v_sигF)
```

The forecast error standard deviations at the locations of the observations are calculated by interpolation from the arrays `z_sигF, u_sигF, v_sигF, psl_sигF, usl_sигF, vsl_sигF, and mix_sигF`. This is done using a call to `intp_sигF()`:

```fortran
CALL intp_sигF(im,jnp,mlev,pres_lev,psl_sигF,usl_sигF, vsl_sигF,z_sигF,u_sигF,v_sигF,mix_sигF, &
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_sигF()` 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()`:

```fortran
call solve4x(maxreg,iregbeg,ireglen,ityplen,npieces, &
            kx_list,lat_list,lon_list,pres_list, &
            sigU_list,sigO_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 \( AI = w^a - w^f \). The \( AI \) are returned by the routine `getAinc()`:

```fortran
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,psi_inc,u_inc,v_inc,z_inc,mix_inc,u,sigF,v,sigF,z,sigF,mix,sigF,ktwanted(ktus), &
             ktwanted(ktusl),ktwanted(ktwv),ktwanted(ktww),ktwanted(ktq),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()`:

```fortran
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()`:

```fortran
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 $u_{\text{sigF}}(i:m,1:jnp,1:mlev)$ and $v_{\text{sigF}}(i:m,1:jnp,1:mlev)$, 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.
- kname0(): Fills in the entries of the arrays stored in the common blocks defined in the header file kttabl.h. Discussed in Section 5.2.
- kname0(): Initializes 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.
- setOEclas(): Determine the set of distinct observation error classes and initialize State I data for observation error standard deviations. Discussed in Section 5.4.
- setOEhCor(): Initialize State I tables for observation error horizontal correlation functions. Discussed in Section 5.5.2.
- setOEvCor(): Initialize State I tables for observation error vertical correlation coefficients. Discussed in Section 5.5.1.
- setFEhCor(): Initialize State I tables for forecast error horizontal correlation functions. Discussed in Section 5.6.2.
- setFEvCor(): Initialize State I tables for forecast error vertical correlation coefficients. Discussed in Section 5.6.1.
- tablFEsigW(): Initialize State I data for model functions for streamfunction and velocity potential forecast error standard deviations $\sigma^\psi$ and $\sigma^\chi$, respectively. Discussed in Section 5.7.
- tablFEalpha(): Initialize State I data for model functions for the geostrophic balance parameters $\alpha_{um}$, $\alpha_{ul}$, $\alpha_{vm}$, and $\alpha_{vl}$. Discussed in Section 5.8.
- bands0(): Initialize parameters used by the conjugate gradient solver conjgr(). Discussed in Section 8.
- kname0(): Initialize the array kname with the icosahedral region names.
- seticos(): Set parameters for the icosahedral decomposition.
- proxel0(): Initialize parameters for the proximity elimination superobbing scheme.
- initInc(): 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 kttabl10 and kttabl0 defined in the header file kttabl.h:

common/kttabl10/ktmvar(ktmax,ktmax)
common/kttabl0/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::

<table>
<thead>
<tr>
<th>#</th>
<th>kt</th>
<th>name</th>
<th>unit</th>
<th>desc</th>
<th>ktmvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>u_</td>
<td>SeaLevel East-West Wind</td>
<td>m/sec</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>v_</td>
<td>Sea Level South-North Wind</td>
<td>m/sec</td>
<td></td>
<td>1 1</td>
</tr>
<tr>
<td>3</td>
<td>p_</td>
<td>Sea Level Pressure</td>
<td>hPa</td>
<td></td>
<td>1 1 1</td>
</tr>
<tr>
<td>4</td>
<td>U_A</td>
<td>Upper Air East-West Wind</td>
<td>m/sec</td>
<td></td>
<td>0 0</td>
</tr>
<tr>
<td>5</td>
<td>U_A</td>
<td>Upper Air South-North Wind</td>
<td>m/sec</td>
<td></td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>6</td>
<td>U_A</td>
<td>Upper Air Geopotential Height</td>
<td>m</td>
<td></td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>7</td>
<td>q</td>
<td>Upper Air Water Vapor Mixing Ratio</td>
<td>g/kg</td>
<td></td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>
```

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 k1 and k2 are one of {1,...,ktmax}, then ktmvar(k1,k2) = 1 whenever the forecast error covariances of k1 and k2 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 occurred.

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 (kxranks), and a CHARACTER data source description array kxdesc. The elements of kxranks(1:ktmax) 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 kttabl0 and kttabl10 defined in the header file kttabl.h:
The variable \texttt{kxtmask} is initialized by \texttt{setGEclas} (Section 5.4).

The values of \texttt{kxclas}, \texttt{kxdesc}, and \texttt{kxrank} are set by calling subroutine \texttt{rdkxtbl()}:

\begin{verbatim}
call rdkxtbl(RC_kx,kxmax,kxclas,kxrank,kxdesc,istat)
\end{verbatim}

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

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

\begin{verbatim}
# kx clas rank desc
10 DROPWSBD -500 Dropwinsonde
11 RADAR_WD -200 Radar-tracked Rawinsonde
12 ROCKTSBD -400 Rocketsonde
13 BALLOON -800 Balloon
14 AIR_ASRD -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 Cl Trk Wind - Wisc El
20 CLDTRKWD -3600 Cl Trk Wind - Wisc E2
21 CLDTRKWD -3700 Cl Trk Wind - Wisc W
22 CLDTRKWD -3800 Cl Trk Wind - Wisc Ocean
24 CLDTRKWD -4000 Cl Trk Wind - IESS East
25 CLDTRKWD -4100 Cl Trk Wind - IESS West
26 CLDTRKWD -4200 Cl Trk Wind - European
27 CLDTRKWD -4300 Cl Trk Wind - Japanese
\end{verbatim}
5.4 Determination of Observation Error Classes and State I data for $\sigma_{ou}$ and $\sigma_{oc}$—setOEclas()

The PSAS constructs State I observation error standard deviation tables using a call to the subroutine setOEclas() (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.

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

```fortran
    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:

```fortran
    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():

```fortran
    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 rdlevelso() 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 rdlevelso(). The section of the resource file read by rdlevelso() 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 $\sigma_{ou}$ and $\sigma_{oc}$: The State I observation error standard deviation tables for $\sigma_{ou}$ and $\sigma_{oc}$ are contained in the arrays sigh and sigoc respectively (Figure 14). These arrays are defined in the module OEclass_tbl, and enter set_OEclas through a USE statement. Both sigh 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, $\sigma_{ou}$ and $\sigma_{oc}$ are defined for only a subset of the observation error levels contained in the array plev_oe. The first vertical levels for which $\sigma_{ou}$ and $\sigma_{oc}$ are defined are specified by INTEGER starting level indices loc_sigh(1:ktmax,1:nOEclas) and loc_sigoc(1:ktmax,1:nOEclas), respectively. The number of vertical levels for which $\sigma_{ou}$ and $\sigma_{oc}$ are defined is stored in the INTEGER arrays leng_sigh(1:ktmax,1:nOEclas) and leng_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($\nu_{ou}$), voecH-c($\nu_{oc}$), and hoecH-c($\rho_{oc}$).

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

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

   voecH_u(1:nOEclas)=''
   loc_sigh(1:ktmax,1:nOEclas)=0
   leng_sigh(1:ktmax,1:nOEclas)=0
   sigh(1:lvmax_oe,1:ktmax,1:nOEclas)=-1.

   hoecH_c(1:nOEclas)=''
   voecH_c(1:nOEclas)=''
   loc_sigoc(1:ktmax,1:nOEclas)=0
   leng_sigoc(1:ktmax,1:nOEclas)=0
   sigoc(1:lvmax_oe,1:ktmax,1:nOEclas)=-1.

   The ‘-1.’ values assigned to the elements of sigh 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)(1:1)\':

   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 $\sigma_{ou}$ and $\sigma_{oc}$ for each data type kt for this observation error class.

   (b) This temporary resource label is used in a call to rdoetbl():
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 `lvmax_oe` 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 1 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)`, `voeH-c(iClass)`, and `hoeH-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 `sigma_u` values are stored in `sigOu(:,kt,iClass)` nonzero values beginning with index `locsigOu(kt,iClass)`. The `sigma_v` values are stored in `sigOc(:,kt,iClass)` nonzero values beginning with index `locsigOc(kt,iClass)`.

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

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

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

iii. If either `locsigOu(kt,iClass) > 0` or `locsigOc(kt,iClass) > 0`, then the observation error class `iClass` has defined values for either `sigma_u` or `sigma_v`, 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(l:kxmax,l:ktmax)` to .true. if `KTclas(kt,iKxclas(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,...,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 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 7.00 7.00 7.00
7.00 7.00
```

The routine `rdoetbl()` interprets this as follows:
• $\sigma_{ou}$ is defined for $u$ and $v$, $\sigma_{oc}$ is not defined.

• Values of upper air zonal wind $\sigma_{ou}$ for 14 vertical levels follow the label $u_{.UprAir.u}$.

• Values of upper air meridional wind $\sigma_{ou}$ 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
  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
  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 27.92 28.28 39.50 30.66
  vCor_HH.u 1
```

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

• $\sigma_{ou}$ is defined for $q$, $h$, $u$ and $v$.

• $\sigma_{oc}$ is not defined.

• Values of upper air zonal wind $\sigma_{ou}$ for 6 vertical levels follow the label $q_{.UprAir.u}$.

• Values of upper air zonal wind $\sigma_{ou}$ for 18 vertical levels follow the label $u_{.UprAir.u}$.

• Values of upper air meridional wind $\sigma_{ou}$ for 18 vertical levels follow the label $v_{.UprAir.u}$.

• Values of upper air geopotential height $\sigma_{ou}$ 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 \texttt{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 $\nu_{ou}$ and $\nu_{oc}$ 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 $\nu_{ou}$ and $\nu_{oc}$ are (Figure 16):

- **n_voeH** (INTEGER): The number of distinct vertical correlation tables needed to compute $\nu_{ou}$ and $\nu_{oc}$ for all the observation error classes determined by the routine set_OEclas().
- **name_voeH** (CHARACTER): An array of names for each vertical correlation coefficient table.
- **type_voeH** (CHARACTER): An array of vertical correlation coefficient table types.
- **desc_voeH** (CHARACTER): An array of short descriptions for each vertical correlation coefficient table.
- **nlev_voeH** (INTEGER): An array containing the number of pressure levels used in each vertical correlation coefficient table.
- **plev_voeH** (REAL): An array containing the pressure levels used to define each vertical correlation coefficient.
- **corr_voeH** (REAL): The set of State I vertical correlation coefficient tables for $\nu_{ou}$ and $\nu_{oc}$.

These data are defined in the module voeh_tbl and are accessed by a USE module statement.

The routine set_OEvCor() (Figure 15) determines the number **n_voeH** of vertical correlation tables required to compute $\nu_{ou}$ and $\nu_{oc}$ for each observation error class, creates indices between **kx** values and the vertical correlation tables contained in **corr_voeH**, and reads from the resource file the values of **name_voeH**, **type_voeH**, **desc_voeH**, **nlev_voeH**, **plev_voeH**, and **corr_voeH**.

Subroutine set_OEvCor() performs the following steps to initialize the State I data for $\nu_{ou}$ and $\nu_{oc}$:

1. Sets the integer variable **nvc** to the number of defined values in the CHARACTER arrays voeh_c and voeh_u ($\text{nvc} \leq 2 * \text{nOEclas}$). The defined values of voeh_c and voeh_u are stored in the CHARACTER array kxvoeh(1:nvc).

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

   ```
   call tabSlist(nvc,kxvoeh,MX_voeH,n_voeH,name_voeH)
   ```

   Subroutine tabSlist() returns the CHARACTER array **name_voeH(1:n_voeH)** of **n_voeH** 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_voeH**:

   (a) For values of **kx** for which $\nu_{oc}$ is defined, assign the appropriate vertical correlation coefficient name:
do kx=1,kxmax
   kc=ixxclas(kx) ! its OE class index
   kxvoecH(kx)=kxclas(kx) ! 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_voeCh)

The output from inxSlist() is the INTEGER index array i_voeCh(1:kxmax). For given kx in {1,...,kxmax}, kxvoecH(kx) = name_voeCh(i_voeCh(kx)).

4. Steps (a-b) above are repeated for v, by storing defined values of voecH_u in kxvoecH, and calling inxSlist():

   call inxSlist(n_voeCh,name_voeCh,kxmax,kxvoecH,i_voeCh)

The output from inxSlist() is the INTEGER index array i_voeCh(1:kxmax). For given kx in {1,...,kxmax}, kxvoecH(kx) = name_voeCh(i_voeCh(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,i),istat)

Example 5.5.1.1: For rawinsonde data (which have only \( \sigma_{ou} \) defined), the appropriate entry in name_voeCh is 'ObeErrvCor.HH-1', and the value of rc_tmp created by setEvCor() is rc_tmp = 'ObsErrvCor.HH-1:'. Below an excerpt is shown from the resource file that is read by rdvctbl():

```
ObsErrvCor.HH-1:
# p 1000 850 700 500 400 300 250 200 150 100 70 50 30 10 5 2 1
100 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 .50 .61 .68 .74 .82 .89 1.
40 .00 .02 .04 .05 .07 .10 .26 .44 .56 .65 .72 .78 1.
30 -.01 -.01 .01 .04 .05 .09 .10 .26 .43 .57 .66 .73 .88 1.
20 -.03 -.01 .02 .02 .04 .04 .11 .26 .44 .56 .65 .72 .78 1.
10 -.02 -.02 -.02 -.02 -.04 -.04 .06 .13 .28 .49 .65 .71 .77 .86 1.
5 -.03 -.02 -.02 -.02 -.04 -.04 .06 .16 .33 .56 .64 .69 .74 .84 1.
2 -.02 -.02 -.02 -.02 -.04 -.04 .06 .18 .34 .56 .64 .69 .74 .84 1.
1 -.01 -.01 -.01 -.01 -.01 -.01 -.01 .01 .01 .01 .01 .01 .01 .01 .01
.4 -.02 -.02 -.02 -.02 -.02 -.02 -.02 -.02 -.02 -.02 -.02 -.02 -.02 -.02 -.02
::
```

41
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 `::`. 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_OEvCor()

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

- `n_hoech` (INTEGER): The number of distinct observation error horizontal correlation functions tables needed to compute $\rho_{oc}$ 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_OEvCor()` (Figure 17) determines the number `n_hoech` of observation error horizontal correlation functions for $\rho_{oc}$ for each observation error class, creates indices between $kk$ 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 $\rho_{oc}$:

\[\text{See the I90 library and its documentation at ftp://niteroi.gsfc.nasa.gov/pub/packages/i90}\]
Figure 16: Data structures containing State I tables for $\nu_{0u}$ and $\nu_{0c}$. 
1. Sets the integer variable nhc to the number of defined values in the CHARACTER array hoecH_c (nhc ≤ 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():

\[
\text{call tabSlist}(\text{nhc}, \text{kxhoecH}, \text{MX_hoeC}, \text{n_hoeC}, \text{name_hoeC})
\]

Subroutine tabSlist() returns the CHARACTER array name_hoeC(1:n_hoeC) of n_voeC 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_hoeC. This is a two-step process:

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

\[
\text{do } kx=1, kxmax \\
\text{kc} = \text{i_kxclas}(kx) \quad \text{! its OE class index} \\
kxhoecH(kx) = \text{hoecH}_c(kc) \quad \text{! its hoecH}_c name \\
\text{end do}
\]

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

\[
\text{call inxSlist}(\text{n_hoeC}, \text{name_hoeC}, \text{kxmax}, \text{kxhoecH}, \text{i_hoeHc})
\]

The output from inxSlist() is the INTEGER index array i_hoeHc(1:kxmax). For given kx one of \{1, ..., kxmax\}, kxhoecH(kx) = name_hoeC(i_hoeHc(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_hoeC. The CHARACTER resource label rc_tmp is formed from name_hoeC(i), and the table is read by calling rdpars():

\[
\text{call rdpars}(\text{rc_tmp}, \text{type_hoeC}(i), \text{desc_hoeC}(i), \text{lVmax}, \text{nlev_hoeC}(i), \text{plev_hoeC}(i,i), \text{MXpar_hc}, \text{npar_hoeC}(i), \text{pars_hoeC}(i,i,i), \text{istat} )
\]

The resulting set of data structures is shown in Figure 18.
Figure 18: Data structures containing State I Data for $\rho_{oc}$. 
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 rdvctbl1() for each value of km:
The vertical correlation model parameters are returned in the array \texttt{corr-vfecH}, with the dimensions of the tables of defined entries given by \texttt{nlev-vfecH}, and the vertical levels for which the parameters are defined by \texttt{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 \texttt{rdvctbl():}

\begin{verbatim}
call rdvctbl( name_vfecQ, type_vfecQ, desc_vfecQ, &
  lvmax_vc, nlev_vfecQ, plev_vfecQ, corr_vfecQ, istat
)
\end{verbatim}

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

### 5.6.2 State I Forecast Error Horizontal Correlations—\texttt{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):
  - \texttt{MX_fecH (INTEGER)}: The maximum number of forecast height/wind error correlations. Defined in the include file \texttt{MX_hfecH.h}. Currently, \texttt{MX_fecH = 3}.
  - \texttt{name_hfecH (CHARACTER)}: An array of names for each forecast height/wind error horizontal correlation function.
  - \texttt{type_hfecH (CHARACTER)}: An array of types for each forecast height/wind error horizontal correlation function.
  - \texttt{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_hfecz` (INTEGER): An array containing the number of pressure levels on which parameters for each forecast height/wind error horizontal correlation function are defined.
- `plev_hfecz` (REAL): An array containing the pressure levels on which parameters for each forecast height/wind error horizontal correlation function are defined.
- `pars_hfecz` (REAL): The set of parameters used in the height/wind forecast error horizontal correlation functions.

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

- State I data for upper-air water vapor forecast error horizontal correlations (Figure 24):
  - `name_hfecz` (CHARACTER): Name of the forecast water vapor mixing ratio error horizontal correlation function.
  - `typshfecz` (CHARACTER): The type of forecast water vapor mixing ratio error horizontal correlation function.
  - `deschfecz` (CHARACTER): A short description of the forecast water vapor mixing ratio error horizontal correlation function coefficient table.
  - `nlev_hfecz` (INTEGER): The number of pressure levels on which parameters for forecast water vapor mixing ratio error horizontal correlation function are defined.
  - `plev_hfecz` (REAL): An array containing the pressure levels on which parameters for forecast water vapor mixing ratio error horizontal correlation function are defined.
  - `pars_hfecz` (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 `hfecz_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_KEhCor()` (Figure 22).

Subroutine `set_KEhCor()` 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_KEhCor()` (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_hfecz` are defined in the module `hfecz_tbl`

   ```fortran
   do km=1,MX_fecz
      call rdpars(name_hfecz(km),type_hfecz(km),desc_hfecz(km), &
                  lmax_hc,nlev_hfecz(km),plev_hfecz(1,km), &
                  MXpar_hc,npar_hfecz(km),pars_hfecz(1,1,km), &
                  isterat
   end do
   ```

   The horizontal correlation function parameter tables are returned in the array `pars_hfecz`, with the dimensions of the tables of defined entries given by `npar_hfecz` and `nlev_hfecz`, and the vertical levels for which the parameters are defined by `plev_hfecz` (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 \texttt{setFEhCor()}.  

\begin{verbatim}
name_hfecH(1:3)
type_hfecH(1:3)
desc_hfecH(1:3)
nlev_hfecH(1:3)
\end{verbatim}

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

\begin{verbatim}
call rdpars(name_hfecQ,type_hfecQ,desc_hfecQ,
& lvmax_hc,nlev_hfecQ,plev_hfecQ,
& MXpar_hc,npar_hfecQ,pars_hfecQ,istat
)
\end{verbatim}

The resource label \texttt{name_hfecQ} is defined in the module \texttt{hfecQ.tbl}. This variable is used by \texttt{rdpars()} to locate and load the model parameter tables. The parameter tables are returned in \texttt{pars_hfecQ(1:npar_hfecQ,1:nlev_hfecQ)}, and the pressure levels on which the parameters are defined in \texttt{plev_hfecQ(1:nlev_hfecQ)}. The name of the horizontal correlation function and a short description are stored in the \texttt{CHARACTER} variables \texttt{type_hfecQ} and \texttt{desc_hfecQ}, respectively (Figure 24).
The State I data for the stream function and velocity potential forecast error standard deviations $\sigma^\psi$ and $\sigma^\chi$ are (Figure 26):

- **FESigW_name (CHARACTER)**: The name of the functions for $\sigma^\psi$ and $\sigma^\chi$.
- **FESigW_type (CHARACTER)**: The type of functions for $\sigma^\psi$ and $\sigma^\chi$.
- **FESigW_desc (CHARACTER)**: A short description of the functions for $\sigma^\psi$ and $\sigma^\chi$.
- **FESigW_nlev (INTEGER)**: The number of pressure levels on which parameters are defined for the functions for $\sigma^\psi$ and $\sigma^\chi$.
- **FESigW_plev (REAL)**: The pressure levels on which parameters are defined for the functions for $\sigma^\psi$ and $\sigma^\chi$.
- **FESigW_npar (INTEGER)**: The number of parameters in the functions for $\sigma^\psi$ and $\sigma^\chi$.
- **FESigW_pars (REAL)**: The set of parameters used in the functions for $\sigma^\psi$ and $\sigma^\chi$.

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

The State I data for $\sigma^\psi$ and $\sigma^\chi$ 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 24: State I data structures mixing ratio forecast error horizontal correlations.

Figure 25: State I data initialization routine tabl.FESigW().

Figure 26: State I data structures stream function and velocity potential forecast error standard deviations.
The input CHARACTER resource label FEsigW_rsrc is defined (with value FEsigW_rsrc = 'FcstErr*Sigma.Wind:') in the module FEsigW_tab1. The routine rdpars() parses the resource file and returns the State I data for \( \sigma^2 \) and \( \sigma^\lambda \). 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 \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \) — tab1.FEalpha()

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

- **FEalpha.name** (CHARACTER): The name of the functions for \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \).
- **FEalpha.type** (CHARACTER): The type of functions for \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \).
- **FEalpha.desc** (CHARACTER): A short description of the functions for \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \).
- **FEalpha.nlev** (INTEGER): The number of pressure levels on which parameters are defined for the functions for \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \).
- **FEalpha.plev** (REAL): The pressure levels on which parameters are defined for the functions for \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \).
- **FEalpha.npar** (INTEGER): The number of parameters in the functions for \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \).
- **FEalpha.pars** (REAL): The set of parameters used in the functions for \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \).

The above data are defined in the module FEalpha.tab1, and are accessed by a USE statement.

The State I data for \( \alpha_{um} \), \( \alpha_{ul} \), \( \alpha_{vm} \), and \( \alpha_{vl} \) are read from the resource file by the routine tab1.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 $\sigma_\psi$ and $\sigma_x$. 
call rdpars(FEalpha_rsrm, FEalpha_type, FEalpha_desc, &
1vmax, FEalpha_nlev, FEalpha_lev, &
FEalpha_Mpar, FEalpha_npar, FEalpha_pars, istat )

The input resource label FEalpha_rsrm passed to rdpars() is defined (with value FEalpha_rsrm = "FctErr*Are::") in the module FEalpha_tabl. The routine rdpars() parses this resource and returns the State I data for $\alpha_{uy}$, $\alpha_{ul}$, $\alpha_{vm}$, and $\alpha_{vl}$. 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 $\alpha_{uy}$, $\alpha_{ul}$, $\alpha_{vm}$, and $\alpha_{vl}$. 
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 $\varphi$.
- Longitude $\lambda$.
- Pressure $p$.
- Time increment from the analysis time $\Delta t$.
- 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,sig0,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*():

55
Table 3: Data box attributes boxes returned by setbox().

<table>
<thead>
<tr>
<th>Element of boxes</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>boxes(1,1,1:nboxes)</td>
<td>Minimum kx value</td>
</tr>
<tr>
<td>boxes(2,1,1:nboxes)</td>
<td>Maximum kx value</td>
</tr>
<tr>
<td>boxes(1,2,1:nboxes)</td>
<td>Minimum kt value</td>
</tr>
<tr>
<td>boxes(2,2,1:nboxes)</td>
<td>Maximum kt value</td>
</tr>
<tr>
<td>boxes(1,3,1:nboxes)</td>
<td>Minimum latitude ( \varphi )</td>
</tr>
<tr>
<td>boxes(2,3,1:nboxes)</td>
<td>Maximum latitude ( \varphi )</td>
</tr>
<tr>
<td>boxes(1,4,1:nboxes)</td>
<td>Minimum longitude ( \lambda )</td>
</tr>
<tr>
<td>boxes(2,4,1:nboxes)</td>
<td>Maximum longitude ( \lambda )</td>
</tr>
<tr>
<td>boxes(1,5,1:nboxes)</td>
<td>Minimum pressure ( p ) (hPa)</td>
</tr>
<tr>
<td>boxes(2,5,1:nboxes)</td>
<td>Maximum pressure ( p ) (hPa)</td>
</tr>
<tr>
<td>boxes(1,6,1:nboxes)</td>
<td>Minimum ( \Delta t ) (minutes)</td>
</tr>
<tr>
<td>boxes(2,6,1:nboxes)</td>
<td>Maximum ( \Delta t ) (minutes)</td>
</tr>
</tbody>
</table>

```
call llboxes ( verbose,luverb,nboxes,boxes,nobs, &
               rlons,rlats,rlevs,kk,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 (\( \sigma_u \)) and sig0 (\( \sigma_0 \)) are scanned for negative values. This is implemented in a call to mark nsig():

```
call mark nsig( nobs,sigU,sig0,kl )
```

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

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

```
call tofront ( nobs,kk,kt,kl,rlats,rlons,rlevs, &
               del,sigU,sig0,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 \texttt{restrict()}. 

Table 4: Data passed into \texttt{restrict()} via its interface. 

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Verbosity control</td>
</tr>
<tr>
<td>lverb</td>
<td>INTEGER</td>
<td>IN</td>
<td>Diagnostic output device</td>
</tr>
<tr>
<td>nobs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Input number of observations</td>
</tr>
<tr>
<td>rlats</td>
<td>REAL(nobs)</td>
<td>INOUT</td>
<td>Latitude ( \varphi )</td>
</tr>
<tr>
<td>rlon</td>
<td>REAL(nobs)</td>
<td>INOUT</td>
<td>Longitude ( \lambda )</td>
</tr>
<tr>
<td>rlevs</td>
<td>REAL(nobs)</td>
<td>INOUT</td>
<td>Pressure level ( p )</td>
</tr>
<tr>
<td>kx</td>
<td>INTEGER(nobs)</td>
<td>INOUT</td>
<td>Data source</td>
</tr>
<tr>
<td>kt</td>
<td>INTEGER(nobs)</td>
<td>INOUT</td>
<td>Data type</td>
</tr>
<tr>
<td>del</td>
<td>REAL(nobs)</td>
<td>INOUT</td>
<td>( w^o - Hw^f )</td>
</tr>
<tr>
<td>sigU</td>
<td>REAL(nobs)</td>
<td>INOUT</td>
<td>( \sigma_{oa} )</td>
</tr>
<tr>
<td>sig0</td>
<td>REAL(nobs)</td>
<td>INOUT</td>
<td>( \sigma_{oc} )</td>
</tr>
<tr>
<td>tstamp</td>
<td>REAL(nobs)</td>
<td>INOUT</td>
<td>Observation time stamp</td>
</tr>
<tr>
<td>nnobs</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Input number of observations</td>
</tr>
</tbody>
</table>
The sorting methodology is explained in Section 3.1.2 (see Figure 5), and is implemented in the subroutine `sort()` (Figure 30):

```fortran
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 `p`, longitude `A`, and level `p`.

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

```fortran
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 `p`, longitude `A`, and level `p` using the routine `typsort()`:

```fortran
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):

---

3In 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expid</td>
<td>CHARACTER*8</td>
<td>IN</td>
<td>Experiment ID tag</td>
</tr>
<tr>
<td>verbose</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Verbosity of diagnostic output</td>
</tr>
<tr>
<td>luverb</td>
<td>INTEGER</td>
<td>IN</td>
<td>Diagnostic output device</td>
</tr>
<tr>
<td>nnobs</td>
<td>INTEGER</td>
<td>INOUT</td>
<td>Number of observations</td>
</tr>
<tr>
<td>rlatls</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Latitude ϕ</td>
</tr>
<tr>
<td>rlonls</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Longitude λ</td>
</tr>
<tr>
<td>rlevs</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Vertical level</td>
</tr>
<tr>
<td>kx</td>
<td>INTEGER(nnobs)</td>
<td>INOUT</td>
<td>Data source</td>
</tr>
<tr>
<td>kt</td>
<td>INTEGER(nnobs)</td>
<td>INOUT</td>
<td>Data type</td>
</tr>
<tr>
<td>del</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$w^o - Hw^f$</td>
</tr>
<tr>
<td>sigU</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_{ou}$</td>
</tr>
<tr>
<td>sigO</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_{oc}$</td>
</tr>
<tr>
<td>sigF</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_f$</td>
</tr>
<tr>
<td>tstamp</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Observation time stamp</td>
</tr>
<tr>
<td>maxreg</td>
<td>INTEGER</td>
<td>IN</td>
<td>Maximum number of regions</td>
</tr>
<tr>
<td>ktnax</td>
<td>INTEGER</td>
<td>IN</td>
<td>Maximum number of data types</td>
</tr>
<tr>
<td>iregbeg</td>
<td>INTEGER(maxreg)</td>
<td>INOUT</td>
<td>Region start index</td>
</tr>
<tr>
<td>ireglen</td>
<td>INTEGER(maxreg)</td>
<td>INOUT</td>
<td>Region length</td>
</tr>
<tr>
<td>ityplen</td>
<td>INTEGER(ktmax,maxreg)</td>
<td>INOUT</td>
<td>$kr/kt$ segment lengths</td>
</tr>
</tbody>
</table>
The subroutine `dupelim` is defined as:

```fortran
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:
    $$|rlev(n) - rlev(n-1)| > TOL$$
  - Their longitudes $\lambda$ differ significantly:
    $$|rlat(n) - rlat(n-1)| > TOL$$
  - Their latitudes $\varphi$ differ significantly:
    $$|rlon(n) - rlon(n-1)| > TOL,$$

  where $TOL$ is a tolerance, currently, set to $10^{-4}$.

- Their data source indices `kx` differ:
  $$kx(n) \neq kx(n-1)$$

- Their data type indices `kt` differ:
  $$kt(n) \neq kt(n-1)$$

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()`:

```fortran
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 \texttt{dupelim()}. 

Table 6: Data passed into \texttt{dupelim()} via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Verbosity of diagnostic output</td>
</tr>
<tr>
<td>luverb</td>
<td>INTEGER</td>
<td>IN</td>
<td>Diagnostic output device</td>
</tr>
<tr>
<td>nnobs</td>
<td>INTEGER</td>
<td>INOUT</td>
<td>Number of observations</td>
</tr>
<tr>
<td>kx</td>
<td>INTEGER(nnobs)</td>
<td>INOUT</td>
<td>Data source</td>
</tr>
<tr>
<td>kt</td>
<td>INTEGER(nnobs)</td>
<td>INOUT</td>
<td>Data type</td>
</tr>
<tr>
<td>kl</td>
<td>LOGICAL(nnobs)</td>
<td>INOUT</td>
<td>Exclusion flag</td>
</tr>
<tr>
<td>rlats</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Latitude $\varphi$</td>
</tr>
<tr>
<td>rlon</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Longitude $\lambda$</td>
</tr>
<tr>
<td>rllevs</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Vertical level</td>
</tr>
<tr>
<td>del</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$w^o - Hw^f$</td>
</tr>
<tr>
<td>sigU</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_{ou}$</td>
</tr>
<tr>
<td>sigO</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_{oc}$</td>
</tr>
<tr>
<td>sigP</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_f$</td>
</tr>
<tr>
<td>tstamp</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Observation time stamp</td>
</tr>
<tr>
<td>maxreg</td>
<td>INTEGER</td>
<td>IN</td>
<td>Maximum number of regions</td>
</tr>
<tr>
<td>iregbeg</td>
<td>INTEGER(maxreg)</td>
<td>INOUT</td>
<td>Region start index</td>
</tr>
<tr>
<td>ireglens</td>
<td>INTEGER(maxreg)</td>
<td>INOUT</td>
<td>Region length</td>
</tr>
<tr>
<td>kmax</td>
<td>INTEGER</td>
<td>IN</td>
<td>Maximum number of data types</td>
</tr>
<tr>
<td>ityplen</td>
<td>INTEGER(kmax, maxreg)</td>
<td>INOUT</td>
<td>$kr/kt$ segment lengths</td>
</tr>
</tbody>
</table>
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):

```fortran
subroutine proxel (verbose,luverb,nnobs,kx,kt,kl,rlats, & rlons,rls,del,sig,Oc,sig,Ou,sigF, & tstamp,maxreg,iregbeg,ireglen, & ktx,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 $(\varphi, \lambda)$. 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 \( x_{\text{obs}}(n_{\text{obs}}) \), \( y_{\text{obs}}(n_{\text{obs}}) \), and \( z_{\text{obs}}(n_{\text{obs}}) \). These variables are used to calculate chordal distances between observations, and for use in determining superob locations.
- INTEGER region lengths workspace array \( i_{\text{reglen}}(\text{maxreg}) \).

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 \( 1 \times 2 \times 3 \times y(z) \):

  \[
  \text{call } 1 \times 2 \times 3 \times y(z)(r \text{lons}, r \text{lats}, n_{\text{obs}}, x_{\text{obs}}, y_{\text{obs}}, z_{\text{obs}}, i_{\text{err}})
  \]

  The Cartesian coordinates \( x, y, \) and \( z \) are returned in \( x_{\text{obs}}(1:n_{\text{obs}}), y_{\text{obs}}(1:n_{\text{obs}}), \) and \( z_{\text{obs}}(1:n_{\text{obs}}) \), respectively.

- The number of surviving observations \( n_{\text{obs}} \) is initialized with the input number of observations \( n_{\text{obs}} \).

- The number of observations eliminated through superobbing \( n_{\text{prox}} \) 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 \( R_{\text{km}} \).
- The observation has the same data type \( k_{\text{t}} \) as the reference observation.
- The observation has not been previously eliminated \( k_{\text{l}} = \text{.TRUE.} \).
- The observation is not already a member of the blacklist; i.e., \( \text{tag} \) is \( \text{.TRUE.} \).

Observations are added to the blacklist by copying their attributes to corresponding blacklist attribute arrays, and incrementing the total number of blacklist observations \( i_{\text{list}} \). 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 \( k_{\text{x} \cdot \text{hi}} \) is determined. Data in the blacklist of type \( k_{\text{x} \cdot \text{hi}} \) are consolidated in a superob through a call to the routine \( \text{prxsob()} \):

  \[
  \text{call prxsob}(i_{\text{err}}, k_{\text{x} \cdot \text{hi}}, d_{\text{id-it}}, \text{tag}, n_{\text{obs}})
  \]

The routine \( \text{prxsob()} \) returns a LOGICAL flag \( d_{\text{id-it}} \) indicating whether or not any observations were eliminated. If \( d_{\text{id-it}} \) is \( \text{.FALSE.} \), no observations were slated for elimination. If \( d_{\text{id-it}} \) is \( \text{.TRUE.} \), observations on the blacklist with \( \text{.FALSE.} \) values in the array \( k_{\text{l} \cdot \text{lst}} \) are to be eliminated. The elimination of observations proceeds, and the region index arrays are updated accordingly. The number \( n_{\text{prox}} \) 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 \( \text{tofront()} \):
Table 7: Black list data structures from proxel.h

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlist</td>
<td>INTEGER</td>
<td>Maximum blacklist size</td>
</tr>
<tr>
<td>idx_lst</td>
<td>INTEGER(nlist)</td>
<td>Index in attribute array</td>
</tr>
<tr>
<td>kx_lst</td>
<td>INTEGER(nlist)</td>
<td>Data source index</td>
</tr>
<tr>
<td>kt_lst</td>
<td>INTEGER(nlist)</td>
<td>Data type index</td>
</tr>
<tr>
<td>ireg_lst</td>
<td>INTEGER(nlist)</td>
<td>Region index</td>
</tr>
<tr>
<td>kl_lst</td>
<td>LOGICAL(nlist)</td>
<td>Elimination flag</td>
</tr>
<tr>
<td>rlats_lst</td>
<td>REAL(nlist)</td>
<td>Latitude $\varphi$</td>
</tr>
<tr>
<td>rlongs_lst</td>
<td>REAL(nlist)</td>
<td>Longitude $\lambda$</td>
</tr>
<tr>
<td>rlevs_lst</td>
<td>REAL(nlist)</td>
<td>Pressure $p$</td>
</tr>
<tr>
<td>del_lst</td>
<td>REAL(nlist)</td>
<td>$w^o - Hw^f$</td>
</tr>
<tr>
<td>sig0u_lst</td>
<td>REAL(nlist)</td>
<td>$\sigma_{0u}$</td>
</tr>
<tr>
<td>sig0c_lst</td>
<td>REAL(nlist)</td>
<td>$\sigma_{0c}$</td>
</tr>
<tr>
<td>sigf_lst</td>
<td>REAL(nlist)</td>
<td>$\sigma_f$</td>
</tr>
<tr>
<td>x_lst</td>
<td>REAL(nlist)</td>
<td>Cartesian $x$ on unit sphere</td>
</tr>
<tr>
<td>y_lst</td>
<td>REAL(nlist)</td>
<td>Cartesian $y$ on unit sphere</td>
</tr>
<tr>
<td>z_lst</td>
<td>REAL(nlist)</td>
<td>Cartesian $z$ on unit sphere</td>
</tr>
</tbody>
</table>

call tofront ( nnobs,kx,kt,kl,rlats,rlons,rlevs, &
               del,sig_0u,sig_0c,sigF,tstamp,nobs )

The routine tofront() counts the number of surviving observations (for which $k1 = .TRUE.$),
returning the count as $nobs$. The attribute arrays are re-arranged so that $k1(1:nobs) = .TRUE.$ and $k1(nobs+1:nobs) = .FALSE.$ 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 superobded 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,sig0,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 $k1$ determines which observations are eliminated. Observations are
Figure 32: Calling tree for \texttt{proxel()}.

Table 8: Data passed into \texttt{proxel()} via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Verbosity of diagnostic output</td>
</tr>
<tr>
<td>luverb</td>
<td>INTEGER</td>
<td>IN</td>
<td>Diagnostic output device</td>
</tr>
<tr>
<td>nnobs</td>
<td>INTEGER</td>
<td>INOUT</td>
<td>Number of observations</td>
</tr>
<tr>
<td>kx</td>
<td>INTEGER(nnobs)</td>
<td>INOUT</td>
<td>Data source</td>
</tr>
<tr>
<td>kt</td>
<td>INTEGER(nnobs)</td>
<td>INOUT</td>
<td>Data type</td>
</tr>
<tr>
<td>kl</td>
<td>LOGICAL(nnobs)</td>
<td>INOUT</td>
<td>Exclusion flag</td>
</tr>
<tr>
<td>rlats</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Latitude $\varphi$</td>
</tr>
<tr>
<td>rlon</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Longitude $\lambda$</td>
</tr>
<tr>
<td>rlevs</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Vertical level</td>
</tr>
<tr>
<td>del</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$w^o - H w^f$</td>
</tr>
<tr>
<td>sig.0u</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_{0u}$</td>
</tr>
<tr>
<td>sig.0c</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_{0c}$</td>
</tr>
<tr>
<td>sigF</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>$\sigma_f$</td>
</tr>
<tr>
<td>tstamp</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Observation time stamp</td>
</tr>
<tr>
<td>maxreg</td>
<td>INTEGER</td>
<td>IN</td>
<td>Maximum number of regions</td>
</tr>
<tr>
<td>iregbeg</td>
<td>INTEGER(maxreg)</td>
<td>INOUT</td>
<td>Region start index</td>
</tr>
<tr>
<td>ireglpen</td>
<td>INTEGER(maxreg)</td>
<td>INOUT</td>
<td>Region length</td>
</tr>
<tr>
<td>ktmx1</td>
<td>INTEGER</td>
<td>IN</td>
<td>Maximum number of data types</td>
</tr>
<tr>
<td>ityplen</td>
<td>INTEGER(ktmax,maxreg)</td>
<td>INOUT</td>
<td>$kr/kt$ segment lengths</td>
</tr>
<tr>
<td>nprox</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Number of eliminated observations</td>
</tr>
</tbody>
</table>
retained if \( k_l = .TRUE. \), and eliminated if \( k_l = .FALSE. \). The number of observations with \( k_l = .TRUE. \) are counted, and this result is stored in the variable \( \text{newnr} \). The array \( k_l \) and the other observation attribute arrays are rearranged so that \( k_l(1:\text{newnr}) = .TRUE. \) and \( k_l(\text{newnr}+1:nobs) = .FALSE. \).

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

The elements of \( \text{rsort} \) are calculated using the following procedure:

1. The number of surviving observations \( \text{newnr} \) is initialized to zero.
2. The array \( k_l(1:nobs) \) is scanned for .TRUE. values. If \( k_l(n) = .TRUE. \), the observation is to be retained in further processing.
   
   \[
   \begin{align*}
   \text{if}( k_l(n) ) & \text{ then} \\
   \text{newnr} & = \text{newnr} + 1 \\
   \text{rsort}(n) & = - \text{float}(\text{newnr}) \\
   \text{else} & \\
   \text{rsort}(n) & = \text{float}(n)
   \end{align*}
   
   \]

   The elements of \( \text{rsort} \) are negative for surviving observations, positive for eliminated observations. The negative elements of \( \text{rsort} \) are in reversed order from the sorting hierarchy the observation attributes possess.

3. The ordering of the elements of \( \text{rsort} \) corresponding to .TRUE. values of \( k_l \) is reversed to the desired order:

   \[
   \begin{align*}
   \text{flip} & = - \text{float}(\text{newnr}) \\
   \text{do } n & = 1, nobs \\
   \text{if}( k_l(n) ) & \text{ rsort}(n) = \text{flip} - \text{rsort}(n) \\
   \text{continue}
   \end{align*}
   
   \]

The array \( \text{rsort} \) is heap-sort indexed by a call to \text{indexxr}():

\[
\text{call INDEXXR ( nobs,rsort,iperm )}
\]

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

The index permutation \( \text{iperm} \) is applied to each of the attribute arrays via calls to \text{permui()}\, \text{permutr()}\, \text{permult()}\:

\[
\begin{align*}
\text{call permui ( } kx,\text{iperm},nobs,kx \ ) \\
\text{call permui ( } kt,\text{iperm},nobs,kt \ ) \\
\text{call permult ( } k1,\text{iperm},nobs,k1 \ ) \\
\text{call permutr ( } rlats,\text{iperm},nobs,rlats \ ) \\
\text{call permutr ( } rlons,\text{iperm},nobs,rlons \ ) \\
\text{call permutr ( } rlevs,\text{iperm},nobs,rlevs \ ) \\
\text{call permutr ( } \text{del, } \text{iperm},nobs,\text{del } \ ) \\
\text{call permutr ( } \text{sigU, } \text{iperm},nobs,\text{sigU } \ ) \\
\text{call permutr ( } \text{sigO, } \text{iperm},nobs,\text{sigO } \ ) \\
\text{call permutr ( } \text{sigF, } \text{iperm},nobs,\text{sigF } \ ) \\
\text{call permutr ( } \text{tstamp, } \text{iperm},nobs,\text{tstamp} \ )
\end{align*}
\]
Table 9: Data passed into \texttt{tofront()} via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nobs</td>
<td>INTEGER</td>
<td>INOUT</td>
<td>Number of observations</td>
</tr>
<tr>
<td>kx</td>
<td>INTEGER(nnobs)</td>
<td>INOUT</td>
<td>Data source</td>
</tr>
<tr>
<td>kt</td>
<td>INTEGER(nnobs)</td>
<td>INOUT</td>
<td>Data type</td>
</tr>
<tr>
<td>kl</td>
<td>LOGICAL(nnobs)</td>
<td>INOUT</td>
<td>Exclusion flag</td>
</tr>
<tr>
<td>rlats</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Latitude ( \varphi )</td>
</tr>
<tr>
<td>rlon</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Longitude ( \lambda )</td>
</tr>
<tr>
<td>rlevs</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Vertical level</td>
</tr>
<tr>
<td>del</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>( w^\circ - H w^f )</td>
</tr>
<tr>
<td>sigU</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>( \sigma_u )</td>
</tr>
<tr>
<td>sigC</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>( \sigma_c )</td>
</tr>
<tr>
<td>sigF</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>( \sigma_f )</td>
</tr>
<tr>
<td>tstamp</td>
<td>REAL(nnobs)</td>
<td>INOUT</td>
<td>Observation time stamp</td>
</tr>
<tr>
<td>newnr</td>
<td>INTEGER</td>
<td>OUT</td>
<td>New number of observations</td>
</tr>
</tbody>
</table>

The final result returned by \texttt{tofront()} is the attribute arrays all sorted so that \( kl(1:\text{newnr}) = \text{.TRUE.} \) and \( kl(\text{newnr}+1:\text{nnobs}) = \text{.FALSE.} \). The two separate segments of the attribute arrays retain the sorting hierarchy shown in Figure 5.
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 $\Sigma_c^o$ and $\Sigma_u^o$ are:

- $\text{sigO} \_\text{list}$ (REAL): Horizontally correlated observation error standard deviation $\Sigma_{oc}$, evaluated at observation locations.
- $\text{sigU} \_\text{list}$ (REAL): Horizontally uncorrelated observation error standard deviation $\sigma_{ou}$, evaluated at observation locations.

The array $\text{sigO} \_\text{list}$ is defined statically in the analysis interface routine (e.g. in $\text{getAIall()}$). The array $\text{sigU} \_\text{list}$ is defined dynamically in the analysis interface routine.

The arrays of $\sigma_{ou}$ and $\sigma_{oc}$ are calculated by interpolation from the State I tables $\text{sigOu}$ and $\text{sigOc}$ defined in the module $\text{OEclass.tbl}$ and initialized by $\text{initRSRC()}$ (see Section 5.4).

The interpolation of $\sigma_{ou}$ and $\sigma_{oc}$ is performed by the subroutine $\text{intp.sigO()}$ (Figure 33):

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

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

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

The routine $\text{intp.sigO()}$ performs the following steps:

1. Checks on variables from $\text{OEclass.tbl}$:
   - The number of levels in the observation error standard deviation tables $\text{nlev.oe} > 0$.
   - The number of observation error classes $\text{nOEcl}as > 0$.

   If either of these conditions are violated, an error message is written to $\text{stdout}$, and execution stops.

2. Two temporary work arrays are allocated:
   - $\text{klev(nobs)}$ (INTEGER), an index table used in the log-linear vertical interpolation.
   - $\text{wlev(nobs)}$ (REAL), a weight table used in the log-linear vertical interpolation.

3. A call to $\text{slogtab()}$ indexes the observation pressure attribute array $\text{rlevs(1:nobs)}$ against the State I table pressure levels $\text{plev.oe(1:nlev.oe)}$: 

68
Figure 33: Calling tree for \texttt{intp\_sigO}. 

\begin{verbatim}
call slogtab(.not.nearest,nlev\_oe,plev\_oe,nobs,rlevs,klev,wlev)

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

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

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

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

\[
\begin{align*}
kc &= i\_kxclas(kxobs(i)) \\
kt &= ktobs(i)
\end{align*}
\]

The index array \texttt{i\_kxclas} is accessed by inclusion of the header file \texttt{kxtabl.h}.

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

\[
\begin{align*}
lc &= loc\_sigOc(kt,kc) \\
ln &= len\_sigOc(kt,kc) \\
le &= lc + ln - 1
\end{align*}
\]

(c) For surface quantities, \( \text{sigC} \) is set directly:

\[
\text{sigC}(i)=\text{sigOc}(lc,kt,kc)
\]

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

\[
\begin{align*}
kl &= klev(i) \\
\text{sigC}(i) &= \text{sigOc}(kl,kt,kc) \\
\text{if}(kl \lt le) \text{ then} \\
& \quad \text{wt}=\text{wlev}(i) \\
& \quad \text{sigC}(i)=\text{sigC}(i) + \text{wt}*(\text{sigOc}(kl+1,kt,kc)-\text{sigC}(i))
\end{align*}
\]

\end{verbatim}
Table 10: Data passed into `intp.sig0()` via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nobs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of observations</td>
</tr>
<tr>
<td>kxobs</td>
<td>INTEGER(nobs)</td>
<td>IN</td>
<td>Data Source</td>
</tr>
<tr>
<td>kobs</td>
<td>INTEGER(nobs)</td>
<td>IN</td>
<td>Data type</td>
</tr>
<tr>
<td>rlevs</td>
<td>REAL(nobs)</td>
<td>IN</td>
<td>Vertical level</td>
</tr>
<tr>
<td>sigC</td>
<td>REAL(nobs)</td>
<td>OUT</td>
<td>$\sigma_{oc}$</td>
</tr>
<tr>
<td>sigU</td>
<td>REAL(nobs)</td>
<td>OUT</td>
<td>$\sigma_{ou}$</td>
</tr>
</tbody>
</table>

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):

```fortran
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:nolev) which contains the values of the levels for the input observations, and alevs(1:nalev) 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 index(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 (nplev < 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
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 $p_{min}$ and $p_{max}$. Valid values are truncated using the following scheme:

   (a) For a given element $olevs(i)$, a truncation granularity $sfIx$ is determined:
   
   \[
   sfIx = \min(smax, rstp \times 10. \times \text{floor}(\log_{10}(olevs(i)))).
   \]

   (b) The value of $olevs(i)$ is rounded to $pfix$, the nearest integral multiple of $sfIx$:
   
   \[
   pfix = sfIx \times \text{nint}(olevs(i)/sfIx).
   \]

   (c) The rounded pressure value $pfix$ is scaled to the current minimum level resolution $s_{min}$. This yields the number $ilevs(1)$ of units of $s_{min}$ in the rounded pressure $pfix$:
   
   \[
   ilevs(1) = \text{nint}(pfix/s_{min}).
   \]

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 1, which is stored in the variable $nwlev$.

5. The set of integer multiples of $s_{min}$ contained in $ilevs(1:nwlev)$ is heap-sort indexed by a call to $\text{indexxii}()$:

   \[
   \text{call indexxii}(nwlev, ilevs, lndex)
   \]

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

6. The set of levels $ilevs(1:nwlev)$ is sorted by applying the permutation $lndex(1:nwlev)$, implemented in a call to $\text{permuti}()$:

   \[
   \text{call permuti}(ilevs, lndex, nwlev, ilevs)
   \]

   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 $s_{min}$ and the array $ilevs$ of level values in units of $s_{min}$. The merged pressure levels $plevs$ are calculated using the formula below:

\[
plevs(1:nplev) = s_{min} \times ilevs(1:nplev)
\]
7.2.2 Latitudinal Entries — merg_lats()

The IMAT structures for the geostrophic balance parameters $a_{\alpha m}$, $a_{\alpha l}$, $a_{\alpha m}$, and $a_{\alpha l}$, and the mass-decoupled forecast wind error standard deviations $\sigma^x$ and $\sigma^y$ 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 $grdlats(ngrdlat)$. 

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nolev</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of levels in table #1</td>
</tr>
<tr>
<td>olevs</td>
<td>REAL(nolev)</td>
<td>IN</td>
<td>Levels in table #1</td>
</tr>
<tr>
<td>nalev</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of levels in table #2</td>
</tr>
<tr>
<td>alevs</td>
<td>REAL(nalev)</td>
<td>IN</td>
<td>Levels in table #2</td>
</tr>
<tr>
<td>mxlev</td>
<td>INTEGER</td>
<td>IN</td>
<td>Maximum number of levels</td>
</tr>
<tr>
<td>nplev</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Number of levels</td>
</tr>
<tr>
<td>plevs</td>
<td>REAL(nplev)</td>
<td>OUT</td>
<td>Merged levels</td>
</tr>
</tbody>
</table>
2. Calculation of the grid point interval $dlat$ from the input number of grid latitudes $ngrdlat$:

   $$ dlat = \frac{180.}{(ngrdlat-1)} $$

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 $MXveclat$:

   $$ res = \frac{180.}{(MXveclat - 1)} $$

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 `merglats()` via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ngrdlat</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid latitudes</td>
</tr>
<tr>
<td>nobslat</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of observation latitudes</td>
</tr>
<tr>
<td>obslats</td>
<td>REAL(nobslat)</td>
<td>IN</td>
<td>Observations' latitudes</td>
</tr>
<tr>
<td>MXveclat</td>
<td>INTEGER</td>
<td>IN</td>
<td>Maximum number of latitudes</td>
</tr>
<tr>
<td>nveclat</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Number of latitudes</td>
</tr>
<tr>
<td>veclats</td>
<td>REAL(nveclat)</td>
<td>OUT</td>
<td>Merged latitudes</td>
</tr>
</tbody>
</table>

7.3 Calculation of Forecast Error Correlation IMAT Tables

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

The state I data for the univariate forecast height and wind error correlation functions \( \rho^H \), \( \rho^X \), and \( \rho^\psi \) 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 /hfecHHO/ hfecHH(MXveclev,nHHtab,MX_fecH)
  common /hfecHHi/ hfecRR(MXveclev,nHHtab,MX_fecH)
  common /hfecHH2/ hfecTT(MXveclev,nHHtab,MX_fecH)
  real HHbeg2,Hcoslim,qxHtbl,qxHtb2
  common /hfecHH3/ HHbeg2,Hcoslim,qxHtbl,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., \( h, \psi, \) or \( \chi \)). The variables in the common block hfecHH3 are described below.

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

  ```
  common/vfecHHO/ 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 \texttt{set\_fecHH()} by inclusion of the include files \texttt{hfecHH.h} (horizontal) and \texttt{vfecHH.h} (vertical). The State I height and wind data defined in Sections 5.6.1 and 5.6.2 are input by \texttt{USE} of the modules \texttt{hfec\_tbl} and \texttt{hfec\_tbl}, respectively.

The IMAT elements are calculated by interpolation from function values calculated using State I tables initialized by \texttt{init\_RSRC()}. These tables are:

- **Horizontal correlation functions:**
  - The horizontal correlation function name \texttt{name\_hfecH}, and type \texttt{type\_hfecH}.
  - The horizontal correlation function has \texttt{npar\_hfecH} parameters, stored in the array \texttt{pars\_hfecH}. These function parameters are level-dependent, and defined on a set of \texttt{nlev\_hfecH} pressure levels stored in the array \texttt{plev\_hfecH}.

These variables are input through \texttt{USE} of the module \texttt{hfec\_tbl}.

- The vertical correlation coefficients are defined in the array \texttt{corr\_vfecH}. The number of levels in each vertical correlation coefficient table are stored in the array \texttt{nlev\_vfecH}, and the level values are stored in \texttt{plev\_vfecH}. These variables are input through \texttt{USE} of the module \texttt{hfec\_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 \texttt{HHinc1} and \texttt{HHinc2} are stored in \texttt{qxHHtb1} and \texttt{qxHHtb2}, respectively.

The routine \texttt{set\_fecHH()} performs the following steps:

1. The values of \texttt{HHbeg2}, \texttt{Hcoslim}, \texttt{qxHHtb1}, and \texttt{qxHHtb2} are calculated using the \texttt{REAL} variables \texttt{HHmx1}, \texttt{HHmx2}, and \texttt{dlim}. The steps in this calculation are:

   (a) Calculation of the value of \texttt{HHmx1}:
   \[
   HHmx1 = 0.036 \times (1. - \cos(6030. / \texttt{rade}))
   \]
   where \texttt{rade} is the Earth’s radius, and enters \texttt{set\_fecHH()} by a \texttt{USE} statement.

   (b) Determination of the value of \texttt{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 \texttt{pars\_hfecH}. This set of elements is scanned to determine their maximum value, which is assigned to the variable \texttt{dlim}:
   \[
   dlim=0.
   \]
   do \texttt{km=1,MX\_fecH}
   do \texttt{lv=1,nlev\_hfecH(km)}
   if(\texttt{pars\_hfecH}(1,lv,km).gt.dlim) \texttt{dlim=pars\_hfecH}(1,lv,km)
   end do
   end do
(c) The value of \( d\text{lim} \) is used to calculate \( HHmx2 \):

\[
HHmx2 = 1 - \cos(d\text{lim}/\text{rade})
\]
\[
HHmx2 = \max(3 \times HHmx1, HHmx2)
\]

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

\[
HHinc1 = HHmx1/n\text{Htb1}
\]
\[
HHinc2 = (HHmx2 - HHmx1)/n\text{Htb2}
\]

(e) The tabulated values of \( 1 - \tau \) for the IMATs are calculated by calling \texttt{intp.\_ctaus()}:

\[
\text{call intp.\_ctaus(nH\text{Htb1}, HHinc1, nH\text{Htb2}, HHinc2, n\text{Htb}, ctaus)}
\]

The tabulated values of \( 1 - \tau \) are returned in \( ctaus(1:n\text{Htb1}+n\text{Htb2}) \).

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

\[
HHbeg2 = ctaus(n\text{Htb1}+1)
\]
\[
Hcoslim = 1 - ctaus(n\text{Htb})
\]

(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 \( km = 1, \text{MX.\_fhecH} \), one value for each of the univariate correlations \( (h, \psi, \chi) \). The following steps occur in the calculation of each correlation function IMAT:

(a) The function type \texttt{type.fhecH(km)} is checked to ensure it is defined. If not, an error message is written to \texttt{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 \texttt{intp.\_hCor()}:

\[
\text{call intp.\_hCor( name.fhecH(km), type.fhecH(km), \&
nlev.hfeciH(km),plev.hfeciH(1,km), \&
MXpar_hc, npar.hfeciH(km),pars.hfeciH(1,1,km), \&
nveclev, pveclev, n\text{Htb}, ctaus, \&
2, MXveclev, hfeciH(1,1,km), \&
hfeciRR(1,1,km), hfeciTT(1,1,km) )}.
\]

The horizontal correlation function IMAT is returned in \( hfeciH(1:n\text{Htb},1:nveclev,km) \), its first derivative with respect to \( 1 - \tau \) in \( hfeciRR(1:n\text{Htb},1:nveclev,km) \), and its second derivative with respect to \( 1 - \tau \) in \( hfeciTT(1:n\text{Htb},1:nveclev,km) \).

(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 \( 0.5 \) since cross-correlation functions in the horizontal are formed by averaging horizontal autocorrelation functions (see [Gaspari et al., 1998] Section 3):

\[
hfeciH(1:nveclev,1:n\text{Htb},km)=.5*hfeciH(1:nveclev,1:n\text{Htb},km)
\]
\[
hfeciRR(1:nveclev,1:n\text{Htb},km)=.5*hfeciRR(1:nveclev,1:n\text{Htb},km)
\]
\[
hfeciTT(1:nveclev,1:n\text{Htb},km)=.5*hfeciTT(1:nveclev,1:n\text{Htb},km)
\]

(d) The vertical correlation IMAT table \texttt{vfeciH} is calculated by a call to \texttt{intp.\_vCor()}:

\[
\text{call intp.\_vCor( name.vfeciH(km), \&
\text{lvmax.\_vc,nlev.vfeciH(km),plev.vfeciH(1,km), \&
corr.vfeciH(1,1,km), \&
\text{MAXveclev,nveclev,pveclev,vfeciH(1,1,km) } )}
\]

The vertical correlation function values for this class are returned in the array \( vfeiH(1:nveclev,1:nveclev,km) \).
Table 13: Forecast Height/Wind Error Horizontal Correlation IMATs

<table>
<thead>
<tr>
<th>IMAT Structure</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>hfecHH(1:nveclev,1:nHHtab,1)</td>
<td>$\rho^h(\tau_{ij}; p_i)$</td>
</tr>
<tr>
<td>hfecHH(1:nveclev,1:nHHtab,2)</td>
<td>$\rho^v(\tau_{ij}; p_i)$</td>
</tr>
<tr>
<td>hfecHH(1:nveclev,1:nHHtab,3)</td>
<td>$\rho^x(\tau_{ij}; p_i)$</td>
</tr>
<tr>
<td>hfecRR(1:nveclev,1:nHHtab,1)</td>
<td>$\rho^h(\tau_{ij}; p_i)$</td>
</tr>
<tr>
<td>hfecRR(1:nveclev,1:nHHtab,2)</td>
<td>$\rho^v(\tau_{ij}; p_i)$</td>
</tr>
<tr>
<td>hfecRR(1:nveclev,1:nHHtab,3)</td>
<td>$\rho^x(\tau_{ij}; p_i)$</td>
</tr>
<tr>
<td>hfecTT(1:nveclev,1:nHHtab,1)</td>
<td>$\rho^h(\tau_{ij}; p_i)$</td>
</tr>
<tr>
<td>hfecTT(1:nveclev,1:nHHtab,2)</td>
<td>$\rho^v(\tau_{ij}; p_i)$</td>
</tr>
<tr>
<td>hfecTT(1:nveclev,1:nHHtab,3)</td>
<td>$\rho^x(\tau_{ij}; p_i)$</td>
</tr>
</tbody>
</table>

(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 vfechDD:

```
do l=1,nveclev
   do k=1,nveclev
      vfechD(k,l,km) = vfechH(k,l,km) * norm_HH(k,km) * norm_DD(l,km)
      vfechDD(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 $\rho^q$. The State II data for $\rho^q$ 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

<table>
<thead>
<tr>
<th>IMAT Structure</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{vfecHH}(1:1;1:1)$</td>
<td>$\frac{\nu^h(p_1,p_1)}{2[\nu^h(p_1,p_1)\rho_h(1:p_1)\nu^h(p_2,p_2)\rho_h(1:p_2)]^{1/2}}$</td>
</tr>
<tr>
<td>$\text{vfecHH}(1:1;1:2)$</td>
<td>$\frac{\nu^e(p_1,p_1)}{2[\nu^e(p_1,p_1)\rho_e(1:p_1)\nu^e(p_2,p_2)\rho_e(1:p_2)]^{1/2}}$</td>
</tr>
<tr>
<td>$\text{vfecHH}(1:1;1:3)$</td>
<td>$\frac{\nu^e(p_1,p_1)}{2[\nu^e(p_1,p_1)\rho_e(1:p_1)\nu^e(p_2,p_2)\rho_e(1:p_2)]^{1/2}}$</td>
</tr>
<tr>
<td>$\text{vfecHD}(1:1;1:1)$</td>
<td>$\frac{\nu^h(p_1,p_1)}{2[\nu^h(p_1,p_1)\rho_h(1:p_1)\nu^h(p_2,p_2)\rho_h(1:p_2)]^{1/2}}$</td>
</tr>
<tr>
<td>$\text{vfecHD}(1:1;1:2)$</td>
<td>$\frac{\nu^e(p_1,p_1)}{2[\nu^e(p_1,p_1)\rho_e(1:p_1)\nu^e(p_2,p_2)\rho_e(1:p_2)]^{1/2}}$</td>
</tr>
<tr>
<td>$\text{vfecHD}(1:1;1:3)$</td>
<td>$\frac{\nu^e(p_1,p_1)}{2[\nu^e(p_1,p_1)\rho_e(1:p_1)\nu^e(p_2,p_2)\rho_e(1:p_2)]^{1/2}}$</td>
</tr>
<tr>
<td>$\text{vfecDD}(1:1;1:1)$</td>
<td>$\frac{\nu^h(p_1,p_1)}{2[\nu^h(p_1,p_1)\rho_h(1:p_1)\nu^h(p_2,p_2)\rho_h(1:p_2)]^{1/2}}$</td>
</tr>
<tr>
<td>$\text{vfecDD}(1:1;1:2)$</td>
<td>$\frac{\nu^e(p_1,p_1)}{2[\nu^e(p_1,p_1)\rho_e(1:p_1)\nu^e(p_2,p_2)\rho_e(1:p_2)]^{1/2}}$</td>
</tr>
<tr>
<td>$\text{vfecDD}(1:1;1:3)$</td>
<td>$\frac{\nu^e(p_1,p_1)}{2[\nu^e(p_1,p_1)\rho_e(1:p_1)\nu^e(p_2,p_2)\rho_e(1:p_2)]^{1/2}}$</td>
</tr>
</tbody>
</table>
Table 15: Forecast Height/Wind Error Correlation Normalization Factors

<table>
<thead>
<tr>
<th>IMAT Structure</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm.HH(1:nveclev,1)</td>
<td>$[\nu^h(p_i, p_i)\rho^h(1; p_i)]^{-1/2}$</td>
</tr>
<tr>
<td>norm.HH(1:nveclev,2)</td>
<td>$[\nu^\psi(p_i, p_i)\rho^\psi(1; p_i)]^{-1/2}$</td>
</tr>
<tr>
<td>norm.HH(1:nveclev,3)</td>
<td>$[\nu^\chi(p_i, p_i)\rho^\chi(1; p_i)]^{-1/2}$</td>
</tr>
<tr>
<td>norm.DD(1:nveclev,1)</td>
<td>$[\nu^h(p_i, p_i)\rho^h(1; p_i)]^{-1/2}$</td>
</tr>
<tr>
<td>norm.DD(1:nveclev,2)</td>
<td>$[\nu^\psi(p_i, p_i)\rho^\psi(1; p_i)]^{-1/2}$</td>
</tr>
<tr>
<td>norm.DD(1:nveclev,3)</td>
<td>$[\nu^\chi(p_i, p_i)\rho^\chi(1; p_i)]^{-1/2}$</td>
</tr>
</tbody>
</table>

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 $1/qqxtb1$ and $0 \leq 1 - \tau < QQbeg2$.
- A coarse-scale range of nQQtb2 values with spacing $1/qqxtb1$ and $QQbeg2 \leq 1 - \tau < 1 - Qcoslim$.

The variables qqxtb1 and qqxtb2 are the inverses of the coarse and fine scale $1 - \tau$ increments QQinci 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/vfecQQO/ 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 setfecQQ() (Figure 36).

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

The routine setfecQQ() performs the following steps:

1. The values of QQbeg2, Qcoslim, qqxtb1, and qqxtb2 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 setfecQQ() 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 QQinci and QQinc2:

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

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

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

80
The tabulated values of $1 - \tau$ are returned in \texttt{ctaus(1:nQQtab1+nQQtab2)}.

(f) Determination of the variables $Qqbeg2$ and $Qcoslim$ from the array \texttt{ctaus}:

\begin{align*}
Qqbeg2 &= \texttt{ctaus(nHHtb1+1)} \\
Qcoslim &= 1.0 - \texttt{ctaus(nHHtab)}
\end{align*}

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

\begin{align*}
qxQtb1 &= 1.0 /Qqinc1 \\
qxQtb2 &= 1.0 /Qqinc2
\end{align*}

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

\begin{verbatim}
call intp_hCor(name_hfecnQQ, &
  type_hfecnQQ,nlev_hfecnQQ,plev_hfecnQQ, &
  MXpar_hc, npar_hfecnQQ,pars_hfecnQQ, &
  nveclev,pveclev,nQQtab, ctaus, &
  0,MXveclev,hfecnQQ,hfecnQQ,hfecnQQ ) .
\end{verbatim}

The argument \texttt{hfecnQQ} is repeated three times in this call. This is to retain a common interface to \texttt{intp_hCor()}. The second and third appearances of \texttt{hfecnQQ} 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 \texttt{intp_hCor()}), only the horizontal correlation function table is returned in \texttt{hfecnQQ(1:nQQtab,1:nveclev)}.

3. The table \texttt{hfecnQQ} is normalized for use in non-separable correlation functions by division by two:

\texttt{hfecnQQ(1:nveclev,1:nQQtab) = 0.5 * hfecnQQ(1:nveclev,1:nQQtab)}

4. The vertical correlation table is calculated by a call to \texttt{intp_vCor()}:

\begin{verbatim}
call intp_vCor('vfecnQQ',lmax_vc,nlev_vfecnQQ,plev_vfecnQQ, &
  corr_vfecnQQ,MXveclev,nveclev,pveclev,vfecnQQ )
\end{verbatim}

The coarse-grained vertical correlation coefficient table is contained in \texttt{corr_vfecnQQ}, which is defined for the \texttt{nlev_vfecnQQ} pressure levels stored in \texttt{plev_vfecnQQ}. The vertical correlation coefficient IMAT is \texttt{vfecnQQ}, which is defined for the IMAT \texttt{nveclev} pressure levels stored in \texttt{pveclev}.

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

The elements of the dense matrices $C^e$ and $C^e$ are obtained by the grid evaluation of the observation error correlation functions. The matrix $C^e$ requires only a set of vertical correlation coefficients, while $C^e$ 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 $C^e$ and $C^e$ are:

- The observation error horizontal correlation function data structures defined in the include file \texttt{hoeckHH.h}:
The horizontal correlation IMAT table is \texttt{hoecHH}, which is organized by pressure level, polarity index \( \tau \), and function class. The variables \texttt{Ocoslim}, \texttt{OObeg2}, \texttt{qxOtb1}, and \texttt{qxOtb2} are discussed later in this section.

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

\begin{verbatim}
common/voecHHO/ voecHH(MXveclev,MXveclev,MX_voecH)
\end{verbatim}

The observation error correlation IMATs are calculated by the routine \texttt{set-oecHH()} (Figure 36). The State I data for the observation error correlations enter \texttt{set-oecHH()} via use of the modules \texttt{hoecH_tbl} (horizontal) and \texttt{voecH_tbl} (vertical). The IMAT vertical levels are input through the include file \texttt{levtabl.h}.

The univariate observation error horizontal correlations are functions of the variable \( 1 - \tau \). Let \( i = 1, \ldots, \text{n_hoecH} \) 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 \( \text{nOOtbl}(i) \) values with spacing \( 1/\text{qxOtb1}(i) \), corresponding to \( 0 \leq 1 - \tau < \text{OObeg2}(i) \).
- A coarse-scale range of \( \text{nHHtb2}(i) \) values with spacing \( 1/\text{qxOtb2}(i) \), corresponding to \( \text{OObeg2}(i) \leq 1 - \tau < 1 - \text{Ocoslim}(i) \).

The routine \texttt{set-oecHH()} performs the following steps:

1. The calculation of the \texttt{n_hoecH} observation error horizontal correlation function tables is implemented in a loop over the index \( i = 1, \text{n_hoecH} \). Each value of \( i \) corresponds to a function class. For each value of \( i \), the following steps are taken:

   \begin{enumerate}
   \item The values of \texttt{OObeg2}(i), \texttt{Ocoslim}(i), \texttt{qxOtb1}(i), and \texttt{qxOtb2}(i) are calculated using the \texttt{REAL} variables \texttt{HHmx1}, \texttt{HHmx2}, \texttt{OOincl}, \texttt{OOinc2}, \texttt{corlen}, and \texttt{dlim}. The steps in this calculation are:
   \begin{enumerate}
   \item Calculation of the value of \texttt{corlen}:
   \begin{verbatim}
   corlen=0.
   do lv=1,nlev_hoecH(i)
      corlen=corlen+pars_hoecH(2,lv,i)
   end do
   corlen=corlen/nlev_hoecH(i)
   \end{verbatim}
   \item Calculation of \texttt{OOincl}:
   \begin{verbatim}
   HHomx1=1.-cos(2.*corlen/rade)
   OOincl=HHomx1/\text{nOOtbl}
   \end{verbatim}
   \item Calculation of \texttt{OOinc2}:
   \begin{verbatim}
   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)
   OOinc2=(HHomx2-HHomx1)/\text{nOOtb2}
   \end{verbatim}
   \end{enumerate}
   \end{enumerate}
iv. Calculation of tabulated $1 - \tau$ values for this observation error class. This is accomplished by a call to \texttt{ctaus()}: 
\begin{verbatim}
call intp_cctaus(nOOtb1,00inc1,nOOtb2,00inc2,nHHotab,ctaus)
\end{verbatim}
The spacings for the fine and coarse $1 - \tau$ ranges are returned in 00inc1 and 00inc2, respectively.

v. Calculation of $\text{qxOtb1}(i)$ and $\text{qxOtb2}(i)$ from 00inc1 and 00inc2:
\begin{verbatim}
\text{qxOtb1}(i)=1./00inc1 
\text{qxOtb2}(i)=1./00inc2
\end{verbatim}

(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 \texttt{intp_hCor()}: 
\begin{verbatim}
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, nveclev, nveclev, ctaus, &
0,MXvcelev,hoecHH(1,1,i),hoecHH(1,1,i), &
hoecHH(1,1,i))
\end{verbatim}
The argument \texttt{hoecHH} is repeated three times in this call. This is done to retain a common interface to \texttt{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 argument list to \texttt{intp_hCor()} (preceding \texttt{MXvcelev}) indicates that derivatives are not calculated.

2. The observation error vertical correlation coefficient IMAT structure \texttt{voecHH} is composed of \texttt{n_voecH} tables, which are calculated by \texttt{n_voecH} calls to \texttt{intp_vCor()}, one for each observation error class:
\begin{verbatim}
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), &
      MXvcelev,nveclev,pveclev, voecHH(1,1,i))
end do
\end{verbatim}
The coarse-grained observation error vertical correlation coefficient tables reside in the array \texttt{corr_voecHH}, and for each class the pressure levels are defined by \texttt{plev_voecH(1:nlev_voecH(i),i)}. These values are interpolated to tables whose dimensions are \texttt{nveclev} by \texttt{pveclev} entries, corresponding to pressure levels contained in the array \texttt{pveclev}. The final product is the IMAT structure \texttt{voecHH}.

7.5 Calculation of IMAT Tables for $\alpha_{um}$, $\alpha_{ul}$, $\alpha_{um}$, and $\alpha_{il}$ — \texttt{imat_alpha()}

The geostrophic balance parameters $\alpha_{um}$, $\alpha_{ul}$, $\alpha_{um}$, and $\alpha_{il}$ that appear in $\Gamma^h$ are stored in the IMAT listed in Table 16. These IMAT tables have dimensions the \texttt{nveclat} latitudes by \texttt{nveclev} pressure levels.

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

The PSAS supports a number of models for $\alpha_{um}$, $\alpha_{ul}$, $\alpha_{um}$, and $\alpha_{il}$. The model used is determined by the CHARACTER tag \texttt{FEalpha_type}, which was read from the resource file during the resource initialization by the subroutine \texttt{tabl_FEalpha()}. The routine \texttt{imat_alpha()}
Table 16: Objects from the module FEalpha_imat

<table>
<thead>
<tr>
<th>IMAT Structure</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aum_imat</td>
<td>$\alpha_{um}$</td>
</tr>
<tr>
<td>Avm_imat</td>
<td>$\alpha_{vm}$</td>
</tr>
<tr>
<td>Aul_imat</td>
<td>$\alpha_{ul}$</td>
</tr>
<tr>
<td>Avl_imat</td>
<td>$\alpha_{vl}$</td>
</tr>
</tbody>
</table>

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 $\alpha$ IMATs:

- case('PSAS:X1'): A call is issued to the internal routine EXP_ALPHAS_model_x1()
- case('EXP-ALPHAS', 'PSAS:X0'): 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 $\sigma^\psi$ and $\sigma^\chi$ — imat_sigW()

The calculation of $\Sigma^\psi$, $\Sigma^\chi$ and their transposes is facilitated by IMAT tables of $\sigma^\psi$ and $\sigma^\chi$ (Table 17). The calculation of the IMATs for $\sigma^\psi$ and $\sigma^\chi$ 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 $\sigma^\psi$ and $\sigma^\chi$ and tabulated values of the $\sigma^\psi$ and $\sigma^\chi$ enter via the module FESigW-tabl.
Table 17: IMATs for $\sigma^\psi$ and $\sigma^x$

<table>
<thead>
<tr>
<th>IMAT Structure</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEsigS_imat</td>
<td>$\sigma^\psi$</td>
</tr>
<tr>
<td>FEsigV_imat</td>
<td>$\sigma^x$</td>
</tr>
</tbody>
</table>

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

The choice of algorithm used to calculate $\sigma^\psi$ and $\sigma^x$ 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 $\sigma^\psi$ and $\sigma^x$ are only level dependent. The calculation is carried out by the internal routine `PSAS11_sigW.a()`. Log-linear vertical interpolation is used to calculate $\sigma^\psi$ and $\sigma^x$ on IMAT levels.

- **case ('PSAS1.1')**: The quantities $\sigma^\psi$ and $\sigma^x$ are only level dependent. The calculation is carried out by the internal routine `PSAS11_sigW()`. Log-linear vertical interpolation is used to calculate $\sigma^\psi$ and $\sigma^x$ on IMAT levels.

- **case ('GEOS/DAS-Cl1')**: Optimal Interpolation option. The calculation is carried out by the internal routine `O1_sigW()`. Log-linear vertical interpolation is used to calculate $\sigma^\psi$ and $\sigma^x$ on IMAT levels.

- **case ('NULL')**: Both $\sigma^\psi$ and $\sigma^x$ 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 $\sigma^{pl}$, the upper-air geopotential height error standard deviation $\sigma^h$, and the upper-air water vapor mixing ratio standard deviation $\sigma^q$. The PSAS reads the $\sigma^h$ and $\sigma^q$ fields from a GrADS binary file, and calculates $\sigma^{pl}$ from $\sigma^h$ using the subroutine `getsigF()` (Figure 39):
subroutine getsigF(iaidim, jaidim, nlevai, plevai, &
    slp_sigsigF, HH_sigsigF, qq_sigsigF)

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

The routine getsigF() performs the following steps to determine $\sigma_{P}^{\text{slf}}$, $\sigma^h$, and $\sigma^q$:

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():

   ```fortran
   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 $\sigma^h$ and $\sigma^q$ are read using calls to the routine rgrads().

4. The geopotential height error standard deviation $\sigma_{P}^{h}$ field corresponding to sea level is read into the array `slp_sigsigF` via a call to rgrads(). The sea-level pressure error standard deviation field $\sigma_{P}^{\text{slf}}$ is calculated by a call to dervsigF_slp():

   ```fortran
   call dervsigF_slp(iaidim, jaidim, slp_sigsigF, slp_sigsigF)
   ```

   The INTEGER arguments `iaidim` and `jaidim` are the horizontal dimensions of the analysis grid. The variable `slp_sigsigF` appears twice in this call; the first argument is the INPUT $\sigma_{P}^{h}$ values, the second the OUTPUT $\sigma_{P}^{\text{slf}}$ 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 $\sigma_{P}^{\text{slf}}$ (`slp_sigsigF`), $\sigma^h$ (`HH_sigsigF`), and $\sigma^q$ (`qq_sigsigF`) are returned to the calling routine through the interface.
Table 18: Data passed into getsigF() via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iaidim</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid latitudes</td>
</tr>
<tr>
<td>jaidim</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid longitudes</td>
</tr>
<tr>
<td>nlevai</td>
<td>REAL(nobslat)</td>
<td>IN</td>
<td>Number of grid vertical levels</td>
</tr>
<tr>
<td>plevai</td>
<td>INTEGER</td>
<td>IN</td>
<td>Pressure levels</td>
</tr>
<tr>
<td>slp.sigF</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>OUT</td>
<td>Forecast error $\sigma_{Psl}$</td>
</tr>
<tr>
<td>HH.sigF</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>OUT</td>
<td>Forecast error $\sigma^h$</td>
</tr>
<tr>
<td>qq.sigF</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>OUT</td>
<td>Forecast error $\sigma^q$</td>
</tr>
</tbody>
</table>

**7.7.2 Gridded Fields of the Elements of $\Sigma^h$ — derivsigF_sld()**

The diagonal sea-level windfield elements of $\Sigma^h$ are the standard deviations of the latitudinal ($\varphi$) and longitudinal ($\lambda$) gradients of the sea-level pressure errors $\epsilon_{P}$. These quantities are currently given by

$$\text{Var}(\partial_{m}\epsilon_{P}) = \text{Var}(\partial_{n}\epsilon_{P}) = \left[\frac{\sigma_{P}^2 \sqrt{\rho^\ast(1)}}{2\Omega \rho a}\right]^2,$$

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

subroutine derivsigF_sld(im,jnp,sigF_pm,sigF_pl,sigF_ps,siF_pl)

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

The calculation entails the vertical interpolation of IMAT values from normDD to the sea-level value pressure value, and use of this interpolated value of $\sqrt{\rho^\ast(1)}$ 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>im</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid latitudes</td>
</tr>
<tr>
<td>jnp</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid longitudes</td>
</tr>
<tr>
<td>sigF_ps</td>
<td>REAL(im,jaidim)</td>
<td>OUT</td>
<td>Forecast error $\sigma^{P_{sl}}$</td>
</tr>
<tr>
<td>sigF_pm</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$\sqrt{\text{Var}(\partial_{\text{m}}e^{P_{sl}})}$</td>
</tr>
<tr>
<td>sigF_pl</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$\sqrt{\text{Var}(\partial_{E}e^{P_{sl}})}$</td>
</tr>
</tbody>
</table>

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

$$
norm = \text{norm\_DD}(lv,kmat\_HGT) + wt \times (\text{norm\_DD}(lv+1,kmat\_HGT) - \text{norm\_DD}(lv,kmat\_HGT))$$

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 `conf i g`.

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

$$
\text{sigF\_pm}(:,:,::) = \text{sigF\_ps}(:,:,::) \times \text{scalar} / \text{norm} \\
\text{sigF\_pl}(:,:,::) = \text{sigF\_pm}(:,:,::)
$$

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

$$
\text{scalar} = 100. / (\text{rhobar} \times 2. \times \text{OMEGA} \times \text{R\_EARTH})
$$

The factor of 100 is a pressure units conversion from hPa to Pa.
7.7.3 Gridded Fields of the Elements of $\Sigma^h$ — dervsigF_upD()

The diagonal upper-air windfield elements of $\Sigma^h$ are the standard deviations of $\varphi$ and $\lambda$ gradients of the upper-air geopotential height forecast errors $\epsilon^h$. These quantities are currently

$$\text{Var}(\partial_m \epsilon^h) = \left(\frac{g a^h \sqrt{\rho^h(1)}}{2 \Omega a}\right)^2,$$

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

```fortran
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 $\sqrt{\rho^h(1)}$ 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 $\sqrt{\rho^h(1)}$ 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 `ulev(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()`:

   ```fortran
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`:

   ```fortran
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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>im</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid latitudes</td>
</tr>
<tr>
<td>np</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid longitudes</td>
</tr>
<tr>
<td>mlev</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid levels</td>
</tr>
<tr>
<td>pres_lev</td>
<td>REAL(mlev)</td>
<td>IN</td>
<td>Number of grid levels</td>
</tr>
<tr>
<td>sigF_Hm</td>
<td>REAL(im,np,mlev)</td>
<td>OUT</td>
<td>$\sqrt{\text{Var}(\partial m \epsilon^h)}$</td>
</tr>
<tr>
<td>sigF_Hl</td>
<td>REAL(im,np,mlev)</td>
<td>OUT</td>
<td>$\sqrt{\text{Var}(\partial l \epsilon^h)}$</td>
</tr>
</tbody>
</table>

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

$$\text{scalar} = G_{\text{EARTH}}/(2.0 C_{\text{OMEGA}} R_{\text{EARTH}})$$

4. Deallocation of the temporary arrays ilev and wlev.

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

7.7.4 Interpolation of Elements of $\Sigma^h$ — intp sigF()

The elements of $\Sigma^h$ corresponding to forecast geopotential height, sea-level pressure, and upper-air water-vapor mixing ratio — $\sigma^h$, $\sigma^p u$, and $\sigma^q$ were read and calculated through a call to getsigF(). Gridded fields of the elements of $\Sigma^h$ corresponding to the wind components were calculated by calling dervsigF_s1D() and dervsigF_upD().

The solution of the Innovation Equation (1) requires the elements of $\Sigma^h$ be evaluated at observation points. The calculation of the analysis increments (AI), Eqn. (2), requires
values of the elements of $\Sigma^h$ at both observation and QEA grid points. The calculation of the elements of $\Sigma^h$ at the observation and QEA grid locations is accomplished via interpolation, implemented in subroutine \texttt{intp\_sigF()} (Figure 42):

\begin{verbatim}
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 )
\end{verbatim}

The arguments to \texttt{intp\_sigF()} are summarized in Table 21.

The routine \texttt{intp\_sigF()} performs interpolation of analysis grid data to locations defined by the attribute vectors \texttt{rlat}, \texttt{rlon}, and \texttt{rlev}. Two types of interpolation are performed:

- Horizontal interpolation of sea-level pressure (\texttt{case(ktps)} below) and winds (\texttt{case(ktpm,ktHp)} below) to sea-level locations listed in \texttt{rlat}, \texttt{rlon}, and \texttt{rlev}. Bilinear interpolation is used.

- Three-dimensional interpolation of upper-air geopotential heights (\texttt{case(ktHH)} below), upper-air winds (\texttt{case(ktHm,ktHl)} below) and upper-air water-vapor mixing ratio (\texttt{case(ktqq)} below) to locations listed in the arrays \texttt{rlat}, \texttt{rlon}, and \texttt{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 \texttt{sigF\_ps}, \texttt{sigF\_pl}, \texttt{sigF\_pm}, \texttt{sigF\_H}, \texttt{sigF\_Hm}, \texttt{sigF\_Hl}, \texttt{sigF\_q}. The \texttt{n\_x} output forecast error standard deviations are returned in the array \texttt{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 \texttt{ilev(mlev,n\_x)} and log-linear interpolation weights \texttt{wlev(mlev,n\_x)}; nearest grid latitude index \texttt{ilat(n\_x)}, and linear interpolation weight \texttt{wlat(n\_x)}; nearest grid longitude index \texttt{ilon(n\_x)} and linear interpolation weight \texttt{wlon(n\_x)}.

   (b) Computation of the nearest grid longitude index \texttt{ilon} and grid latitude index \texttt{ilat} for each output grid location:

   \begin{verbatim}
   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
   \end{verbatim}

   The output grid location referenced by index $i$ has longitude $\lambda$ between $ilon(i) \cdot dlon - 180.$ and $(ilon(i) + 1) \cdot dlon - 180.$, and latitude $\varphi$ between $ilat(i) \cdot dlat - 90.$ and $(ilat(i) + 1) \cdot dlat - 90.$.

   (c) Computation of the bilinear horizontal interpolation weights for each output grid location:

   \begin{verbatim}
   do i=1,n\_x
     wlon(i)=(rlon(i)+180.)/dlon - ilon(i) 
     wlat(i)=(rlat(i)+90.)/dlat - (ilat(i)-1)
   end do
   \end{verbatim}
(d) Index the output grid pressure level values \( rlev(1:n_x) \) against the grid pressure levels \( \text{pres}_\text{lev}(1:\text{mlev}) \) by calling \text{slogtab():}

\[
\text{call } \text{slogtab}(\text{not.nearest},\text{mlev},\text{pres}_\text{lev},\text{n}_x,\text{rlev},\text{ilev},\text{wlev})
\]

The index corresponding to the higher pressure level is returned in \( \text{ilev}(1:\text{mlev},1:\text{n}_x) \). The log-linear interpolation weights in are returned in \( \text{wlev}(1:\text{mlev},1:\text{n}_x) \).

2. The interpolation is carried out as a loop over the \( \text{kr}/\text{kt} \) segments in the output grid vector. The interpolation steps are:

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

\[
\begin{align*}
\text{lc} &= \text{kr}_\text{loc}(\text{kr}) + \text{kt}_\text{loc}(\text{kt},\text{kr}) \\
\text{ln} &= \text{kt}_\text{len}(\text{kt},\text{kr}) \\
\text{le} &= \text{lc} + \text{ln} - 1
\end{align*}
\]

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

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

\[
\begin{align*}
\text{do } i &= \text{lc},\text{le} \\
\text{sigF}(i) &= \text{bilinear}_\text{(sigF_ps,ilon(i),wlon(i),ilat(i),wlat(i))}
\end{align*}
\]

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

\[
\begin{align*}
\text{do } i &= \text{lc},\text{le} \\
\text{sigF}(i) &= \text{bilinear}_\text{(sigF_pm,ilon(i),wlon(i),ilat(i),wlat(i))}
\end{align*}
\]

Note that this assumes that \( \text{sigF_pl} = \text{sigF_pm} \), which is currently what is produced by \text{dervsigF_s1D()} (see Section 7.7.2).

- \text{case(ktHH)}: Upper-air geopotential heights. This is horizontal bilinear interpolation followed by vertical log-linear interpolation. The vertical index array \( \text{ilev} \) determines which grid levels lie immediately above and below the output grid point. Bilinear interpolation is used to calculate the values of \( \text{sigF} \) 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 \( \text{wlev} \):

\[
\begin{align*}
\text{do } i &= \text{lc},\text{le} \\
\text{sigF}(i) &= \text{bilinear}_\text{(sigF_H(1,1,ilev(i)), & ilon(i),wlon(i),ilat(i),wlat(i))} \\
\text{if(ilev(i).lt.mlev) then} \\
\text{sigm} &= \text{bilinear}_\text{(sigF_H(1,1,ilev(i)+1), & ilon(i),wlon(i),ilat(i),wlat(i))} \\
\text{sigF}(i) &= \text{sigF}(i) + \text{wlev}(i) \ast (\text{sigm} - \text{sigF}(i))
\end{align*}
\]

- \text{case(ktHm,ktHl)}: The upper-air winds. Both fields are calculated from gridded values of \( \text{sigF_Hm} \). This is compatible with the current version of \text{dervsigF_upD()} (see Section 7.7.3).

- \text{case(ktqq)}: The upper-air water vapor mixing ratio. Segments of \( \text{sigF} \) corresponding to \( \sigma^\text{q} \) are calculated using the gridded values of \( \sigma^\text{q} \) stored in the array \( \text{sigF_q} \).

3. Upon completion of the interpolation, the temporary arrays \( \text{ilev, wlev, ilat, wlat, ilon, and wlon} \) are deallocated.
Table 21: Data passed into `intp_sigF()` via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>im</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid latitudes</td>
</tr>
<tr>
<td>jnp</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid longitudes</td>
</tr>
<tr>
<td>mlev</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of grid levels</td>
</tr>
<tr>
<td>pres.lev</td>
<td>REAL(mlev)</td>
<td>IN</td>
<td>Number of grid levels</td>
</tr>
<tr>
<td>sigF_ps</td>
<td>REAL(im,jaidim)</td>
<td>OUT</td>
<td>Forecast error ( \sigma_{p})</td>
</tr>
<tr>
<td>sigF_pm</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>( \sqrt{\text{Var}(\partial_m e_{p})} )</td>
</tr>
<tr>
<td>sigF_pl</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>( \sqrt{\text{Var}(\partial_t e_{p})} )</td>
</tr>
<tr>
<td>sigF_H</td>
<td>REAL(im,jnp, mlev)</td>
<td>OUT</td>
<td>Forecast error ( \sigma_{h} )</td>
</tr>
<tr>
<td>sigF_Hm</td>
<td>REAL(im,jnp, mlev)</td>
<td>OUT</td>
<td>( \sqrt{\text{Var}(\partial_m e_{h})} )</td>
</tr>
<tr>
<td>sigF_Hl</td>
<td>REAL(im,jnp, mlev)</td>
<td>OUT</td>
<td>( \sqrt{\text{Var}(\partial_t e_{h})} )</td>
</tr>
<tr>
<td>sigF_q</td>
<td>REAL(im,jnp, mlev)</td>
<td>OUT</td>
<td>( \sigma_{q} )</td>
</tr>
<tr>
<td>n.kr</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>kr.loc</td>
<td>INTEGER(n.kr)</td>
<td>IN</td>
<td>Region Starting Indices</td>
</tr>
<tr>
<td>kr.len</td>
<td>INTEGER(n.kr)</td>
<td>IN</td>
<td>Region Lengths</td>
</tr>
<tr>
<td>n.kt</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Data types</td>
</tr>
<tr>
<td>kt.loc</td>
<td>INTEGER(n.kt, n.kr)</td>
<td>IN</td>
<td>Starting Indices of kr/kt segments</td>
</tr>
<tr>
<td>kt.len</td>
<td>INTEGER(n.kt, n.kr)</td>
<td>IN</td>
<td>Length of kr/kt segments</td>
</tr>
<tr>
<td>n.x</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of output grid points</td>
</tr>
<tr>
<td>rlat</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>Output grid latitude ( \varphi )</td>
</tr>
<tr>
<td>rlon</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>Output grid longitude ( \lambda )</td>
</tr>
<tr>
<td>rlev</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>Output grid pressure (hPa)</td>
</tr>
<tr>
<td>sigF</td>
<td>REAL(n.x)</td>
<td>OUT</td>
<td>Elements of ( \Sigma_{h} ) at output grid locations</td>
</tr>
</tbody>
</table>

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 \( x \) 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():

```fortran
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 \( \hat{e}_r \).
- `ql_x(nobs) and ql_y(nobs)`: Cartesian components of the longitudinal unit vector \( \hat{e}_l \).
- `qm_x(nobs), qm_y(nobs), and qm_z(nobs)`: Cartesian components of the meridional unit vector \( \hat{e}_m \).
- `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 \( \tau_{ij} \) 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 (\(\hat{e}_r, \hat{e}_\theta, \hat{e}_\phi\)) through a call to ll2qvec():

```fortran
  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():

```fortran
  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():

```fortran
  call slogtab(roundoff, nveclev, pveclev, nobs, rlevX, ktab, wtab)
```

• Determine the polarity index \(\tau_{ij}\) level IMAT indices \(jtab(1:nobs)\) and linear interpolation weights \(vtab(1:nobs)\) through a call to the routine slintab():

```fortran
  call slintab(roundoff, nveclat, veclats, nobs, rlatX, jtab, vtab)
```

The conjugate gradient solution \(x\) \((Xvec(1:nobs, 1:nvecs))\) is determined by calling conjgr():

```fortran
  call conjgr(cgverb(nbandcg), n_kr, kr_loc, kr_len, kmax, kt_loc, &
  kr_len, nobs, kXX, kXX, ktab, jtab, sigU, sigG, 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 \((HP^TH^T + R)x\) is performed by a call to the routine \(opMx()\).

• 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 \((HP^TH^T + R)x\) is performed by a call to the routine \(opMx()\).

• 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 \((HP^TH^T + R)x\) is performed by a call to the Basic Linear Algebra Software (BLAS) routine \(SPPMV()\). 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 \(opMx()\), 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 \( \text{solve4x()} \).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_kr</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>kr_loc</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Region Starting Indices</td>
</tr>
<tr>
<td>kr_len</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Region Lengths</td>
</tr>
<tr>
<td>kt_len</td>
<td>INTEGER(ktmax,n_kr)</td>
<td>IN</td>
<td>Length of kr/kt Segments</td>
</tr>
<tr>
<td>nnobs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Innovations</td>
</tr>
<tr>
<td>kxX</td>
<td>INTEGER(nnobs)</td>
<td>IN</td>
<td>Instrument Index ( kx )</td>
</tr>
<tr>
<td>rlatX</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Latitude ( \varphi )</td>
</tr>
<tr>
<td>rlonX</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Longitude ( \lambda )</td>
</tr>
<tr>
<td>rlevX</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Pressure ( \text{hPa} )</td>
</tr>
<tr>
<td>sigU</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Obs. Error ( \sigma_u )</td>
</tr>
<tr>
<td>sigO</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Obs. Error ( \sigma_o )</td>
</tr>
<tr>
<td>sigF</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Forecast Error ( \sigma_f )</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Vectors</td>
</tr>
<tr>
<td>ldD</td>
<td>INTEGER</td>
<td>IN</td>
<td>Innovation Vector Leading dimension</td>
</tr>
<tr>
<td>Dels</td>
<td>REAL(ldD,nvecs)</td>
<td>IN</td>
<td>Innovations ( \mathbf{w}^o - H\mathbf{w}^T )</td>
</tr>
<tr>
<td>ldX</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of ( \mathbf{xvec} ) (( x ))</td>
</tr>
<tr>
<td>Xvec</td>
<td>REAL(ldX,nvecs)</td>
<td>IN</td>
<td>Intermediate vector (( x ))</td>
</tr>
</tbody>
</table>

8.3 Software Implementation of \((HP^TH^T + R)x\) — \( \text{op.Mx()} \)

The matrix-vector multiplication \((HP^TH^T + R)x\) for the global solver and preconditioning level 2 is performed by the routine \( \text{op.Mx()} \). The calling tree for \( \text{op.Mx()} \) is illustrated in Figure 45, and its interface is presented below:

```
subroutine op_Mx(kind_mat(kind_coy,  &
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, q1_x, q1_y, &
nvecs, &
ldx, x,  &
ldCx,Cx  )
```

A description of the arguments to \( \text{op.Mx()} \) is given in Table 23. The routine \( \text{op.Mx()} \) computes the elements of \( HP^TH^T + R \) and calculates the product \((HP^TH^T + R)x\). The vector \( x \) and the product \((HP^TH^T + R)x\) 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 \((HP^TH^T + R)x\) 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 \texttt{covOOxpy()}: 

\begin{verbatim}
call covOOxpy_(ldx, x, ldCx, Cx)
\end{verbatim}

The calculation of $R_{Ux}$ is described below:

1. Calculate $T_x = \Sigma^o x$ using a call to \texttt{mv.diag()}: 

\begin{verbatim}
call mv.diag(n_kr,kr_loc,kr_len,n_kt,kt_loc,kt_len, & 
  n_x,sigU,nvecs,ldx,x,n_x,Tx )
\end{verbatim}

2. Set $T_y(1:n_x,1:nvecs) = 0$. Calculate $T_y = C^o U_x$ using a call to \texttt{sym.Cxpy()}: 

\begin{verbatim}
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 )
\end{verbatim}

3. Set $T_x(1:n_x,1:nvecs) = 0$. Calculate $T_x = \Sigma^o U_x$ using a call to \texttt{mv.diag()}: 

\begin{verbatim}
call mv.diag(n_kr,kr_loc,kr_len,n_kt,kt_loc,kt_len, & 
  n_x,sigU,nvecs,x,n_x,Tx )
\end{verbatim}

This result is added to the output array $y$:

\begin{align*}
y(1:n_x,1:nvecs) &= y(1:n_x,1:nvecs) + T_x(1:n_x,1:nvecs) 
\end{align*}

The calculation and application of $R_C$ follows a similar scheme:

1. Calculate $T_x = \Sigma^c x$ using a call to \texttt{mv.diag()}: 

\begin{verbatim}
call mv.diag(n_kr,kr_loc,kr_len,n_kt,kt_loc,kt_len, & 
  n_x,sigC,nvecs,ldx,x,n_x,Tx )
\end{verbatim}

2. Set $T_y(1:n_x,1:nvecs) = 0$. Calculate $T_y = C^c U_x$ using a call to \texttt{sym.Cxpy()}: 

\begin{verbatim}
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 )
\end{verbatim}

3. Set $T_x(1:n_x,1:nvecs) = 0$. Calculate $T_x = \Sigma^c U_x$ using a call to \texttt{mv.diag()}: 

\begin{verbatim}
call mv.diag(n_kr,kr_loc,kr_len,n_kt,kt_loc,kt_len, & 
  n_x,sigC,nvecs,x,n_x,Tx )
\end{verbatim}

This result is added to the output array $y$:

\begin{align*}
y(1:n_x,1:nvecs) &= y(1:n_x,1:nvecs) + T_x(1:n_x,1:nvecs) 
\end{align*}
The Forecast Error Covariance Operator. The operator $H^T \Sigma^h H^T$ is calculated using the factored-operator expansion Eqn. (7) using the internal procedure *covFFxpy*():

```plaintext
call covFFxpy_ (ldx, x, ldCx, Cx)
```

The product $\Gamma_x^h \Sigma_x^h C_x^h \Sigma_x^h \Gamma_x^h x$ is calculated as follows:

1. Initialize the temporary array $T_x(1:n_x, 1:nvecs)$ to zero. Calculate $T_x = \Gamma_x^h x$ by calling the routine *aj_Alf()*:
   ```plaintext
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 $T_y = \Sigma_y^h x$ by calling the routine *mv_diag()*:
   ```plaintext
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 $T_x(1:n_x, 1:nvecs) = 0$. Calculate $T_x = C_x^h T_y$ by calling the routine *sym_Cxpy()*:
   ```plaintext
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, q1_x, q1_y, &
               nvecs, n_x, Ty, n_x, Tx, 1, istat)
```

4. Calculate $T_y = \Sigma_y^h x$ by calling the routine *mv_diag()*:
   ```plaintext
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 $T_x(1:n_x, 1:nvecs) = 0$. Calculate $T_x = \Gamma_x^h T_y$ by calling the routine *mv_Alf()*:
   ```plaintext
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 $\Gamma_x^h \Sigma_x^h C_x^h \Sigma_x^h \Gamma_x^h x$ is returned $T_x(1:n_x, 1:nvecs)$, which is added to the result $y$ stored in $y(1:n_x, 1:nvecs)$:

```plaintext
y(1:n_x, 1:nvecs) = y(1:n_x, 1:nvecs) + T_x(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 $x$ 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 $\sigma^u$ and $\sigma^x$ 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 $\Gamma^\psi \Sigma^\psi C^\psi \Sigma^\psi \Gamma^\psi T x$ is calculated for both the upper-air and surface winds as follows:

1. Set $Tx(1:n_x, 1:nvecs) = 0$.
2. Initialize the wind error streamfunction standard deviation temporary array $\text{sigW}(1:n_x) = 0$.
   Fill in the necessary elements of $\text{sigW}$ with their respective values of $\sigma^\psi$ obtained from the indirect matrix structure $\text{FEsigS_imat}$. This is implemented as a call to the routine $\text{getivec}()$.

   ```
call getivec(MXveclev, MXveclat, FEsigS_imat, n_x, kr, kr_loc, kr_len, n_kt, kt_loc, kt_lenW, &
n_x, ktab, jtab, sigW )
```
3. Calculate $Tx = \Gamma^\psi T x$ by calling the adjoint routine $\text{aj_Bet}()$:

   ```
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 $Ty = \Sigma^\psi Tx$ by calling $\text{mv_diag}()$:

   ```
call mv_diag(n_kr, kr_loc, kr_len, n_kt, kt_loc, kt_lenW, &
n_x, sigW, nvecs, n_x, Ty )
```
5. Set $Tx(1:n_x, 1:nvecs) = 0$. Calculate $Tx = C^\psi Ty$. This is accomplished by calling the routine $\text{sym_Cxpy}()$:

   ```
call sym_Cxpy(kind_mat, kind_covS, sparse, &
n_kr, kr_loc, kr_len, n_kt, kt_loc, kt_lenW, &
n_x, kr, ks, ktab, &
qr_x, qr_y, qr_z, qr_m, qr_m, qr_l, qr_l, &
nvecs, n_x, Ty, n_x, Tx, istat )
```
6. Calculate $Ty = \Sigma^\psi Tx$ by calling the routine $\text{mv_diag}()$:

   ```
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 $Tx(1:n_x, 1:nvecs) = 0$. Calculate $Tx = \Gamma^\psi Ty$ by calling the routine $\text{mv_Bet}()$:

   ```
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 $\Gamma^\psi \Sigma^\psi C^\psi \Sigma^\psi \Gamma^\psi T x$ is stored in the array $Tx(1:n_x, 1:nvecs)$, which is added to the result $y$ stored in $y(1:n_x, 1:nvecs)$.

100
The final error covariance operator term \( \Gamma^{x} \Sigma_{x}^{y} \Gamma^{x}^{T} x \) is calculated for both the upper-air and surface windfields as follows:

1. Set \( T_{x}(1:n_{x},1:n_{vecs}) = 0 \).
2. Initialize the wind error velocity potential standard deviation temporary array \( \text{sigW}(1:n_{x}) = 0 \). Fill in the necessary elements of \( \text{sigW} \) with their respective values of \( \sigma^{x} \) obtained from the indirect matrix structure \( \text{FESigV.imat} \). This is implemented as a call to the routine \( \text{getivec()}. \)

\[
\text{call getivec(MXveclev,MXveclat,FESigV.imat,}
\text{&
\quad n_{kr},kr_{loc},kr_{len}, n_{kt},kt_{loc},kt_{lenW}, &}
\text{&
\quad n_{x},ktab,jtab,sigW)
}\]

3. Calculate \( T_{x} = \Gamma^{x} T_{x} \) by calling the adjoint routine \( \text{aj.Gam()}: \)

\[
\text{call aj.Gam(n_{kr},kr_{loc},kr_{len}, n_{kt},kt_{loc},kt_{lenW}, &}
\text{n_{x},ktab,jtab,nvecs, idx, x, n_{x}, Tx)
}\]

4. Calculate \( T_{y} = \Sigma_{x}^{y} T_{x} \) by calling \( \text{mv.diag()}: \)

\[
\text{call mv.diag(n_{kr},kr_{loc},kr_{len}, n_{kt},kt_{loc},kt_{lenW}, &}
\text{n_{x},sigW, nvecs, n_{x}, Tx, n_{x}, Ty)
}\]

5. Set \( T_{x}(1:n_{x},1:n_{vecs}) = 0 \). Calculate \( T_{x} = C^{x} T_{y} \). This is accomplished by calling the routine \( \text{sym.Cxpy()}: \)

\[
\text{call sym.Cxpy(kind_mat,kind_covV, sparse, &}
\text{n_{kr},kr_{loc},kr_{len}, n_{kt},kt_{loc},kt_{lenW}, &}
\text{n_{x}, kr, ks, ktab, &}
\text{qr-x, qr-y, qr-z, qm-x, qm-y, qm-z, ql-x, ql-y, &}
\text{nvecs, n_{x}, Ty, n_{x}, Tx, istat)
}\]

6. Calculate \( T_{y} = \Sigma_{x}^{y} T_{x} \) by calling the routine \( \text{mv.diag()}: \)

\[
\text{call mv.diag(n_{kr},kr_{loc},kr_{len}, n_{kt},kt_{loc},kt_{lenW}, &}
\text{n_{x},sigW, nvecs, n_{x}, Tx, n_{x}, Ty)
}\]

7. Set the work array \( T_{x}(1:n_{x},1:n_{vecs}) = 0 \). Calculate \( T_{x} = \Gamma^{x} T_{y} \) by calling the routine \( \text{mv.Gam()}: \)

\[
\text{call mv.Gam(n_{kr},kr_{loc},kr_{len}, n_{kt},kt_{loc},kt_{lenW}, &}
\text{n_{x},ktab,jtab,nvecs, n_{x}, Ty, n_{x}, Tx)
}\]

The term \( \Gamma^{x} \Sigma_{x}^{y} \Sigma_{x}^{y} \Gamma^{x}^{T} x \) is stored in the array \( T_{x}(1:n_{x},1:n_{vecs}) \), which is added to the result \( y \) stored in \( y(1:n_{x},1:n_{vecs}) \):

\[
y(1:n_{x},1:n_{vecs}) = y(1:n_{x},1:n_{vecs}) + T_{x}(1:n_{x},1:n_{vecs})
\]

This completes the calculation and application of the Innovation operator. The final step is to deallocate the temporary arrays \( T_{x}, T_{y}, kt_{lenW}, \) and \( \text{sigW} \) and return.
Table 23: Summary of arguments to op_Mx().

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind_mat</td>
<td>INTEGER</td>
<td>IN</td>
<td>Matrix Type</td>
</tr>
<tr>
<td>kind_cov</td>
<td>INTEGER</td>
<td>IN</td>
<td>Covariance Type</td>
</tr>
<tr>
<td>n_kr</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>kr_loc</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Index of Region Start</td>
</tr>
<tr>
<td>kr_len</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Region Lengths</td>
</tr>
<tr>
<td>n_kt</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Data Types</td>
</tr>
<tr>
<td>kt_loc</td>
<td>INTEGER(n_kt,n_kr)</td>
<td>IN</td>
<td>Start of each kt block for each region</td>
</tr>
<tr>
<td>kt_len</td>
<td>INTEGER(n_kt,n_kr)</td>
<td>IN</td>
<td>Length of each kt block for each region</td>
</tr>
<tr>
<td>n_x</td>
<td>INTEGER</td>
<td>IN</td>
<td>Dimension of Attribute Arrays</td>
</tr>
<tr>
<td>kx</td>
<td>INTEGER(n_x)</td>
<td>IN</td>
<td>Data Source Index</td>
</tr>
<tr>
<td>ks</td>
<td>INTEGER(n_x)</td>
<td>IN</td>
<td>Sounding Index</td>
</tr>
<tr>
<td>ktab</td>
<td>INTEGER(n_x)</td>
<td>IN</td>
<td>Level Index for Look-Up Tables</td>
</tr>
<tr>
<td>jtab</td>
<td>INTEGER(n_x)</td>
<td>IN</td>
<td>Latitude Index for Look-Up Tables</td>
</tr>
<tr>
<td>sigU</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$\sigma_{ou}$</td>
</tr>
<tr>
<td>sigC</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$\sigma_{oc}$</td>
</tr>
<tr>
<td>sigF</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>Forecast Error $\sigma$</td>
</tr>
<tr>
<td>qr_x</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qr_y</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qr_z</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qm_x</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>qm_y</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>qm_z</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>q1_x</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_1$</td>
</tr>
<tr>
<td>q1_y</td>
<td>REAL(n_x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_1$</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of vectors</td>
</tr>
<tr>
<td>ldx</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $x$</td>
</tr>
<tr>
<td>x</td>
<td>REAL(ldx,nvecs)</td>
<td>IN</td>
<td>Input vectors $x$</td>
</tr>
<tr>
<td>ldCx</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $Cx$</td>
</tr>
<tr>
<td>Cx</td>
<td>REAL(ldy,nvecs)</td>
<td>OUT</td>
<td>Output vectors $(HP^T H^T + R)x$</td>
</tr>
</tbody>
</table>
Figure 45: Calling tree for \texttt{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) \( \mathbf{xvec} \) is transformed to state space using Eqn. (2) to produce \( \mathbf{AI} (\mathbf{w}^a - \mathbf{w}^f) \). 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.

```fortran
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, 
   uussigF, vssigF, slpsigF, 
   uussigF, 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 \( \mathbf{usai} (u^d), \mathbf{vsai} (v^d), \mathbf{slpai} (p^d), \mathbf{uuai} (u), \mathbf{vvai} (v), \mathbf{HHai} (h), \) and \( \mathbf{qqai} (q) \). The input LOGICAL variables \( \text{want-slu} \), \( \text{want-slv} \), \( \text{want-slp} \), \( \text{want-uwnd} \), \( \text{want_vwnd} \), \( \text{want_hght} \), and \( \text{want_mixr} \) determine the variables for which AI are calculated. These variables were initialized from the resource file by the routine iniainc(). For example, if \( \text{want-slu} = \text{.TRUE.} \) then AI for \( (p^d) \) are calculated. If \( \text{want-slu} = \text{.FALSE.} \) then AI for \( (p^d) \) 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 \( \mathbf{xvec} \) 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 \( x \) 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 \((\varphi, \lambda)\) (Figure 47). The ordering of the variables in the G-vector is defined by the following rules:
Figure 10: 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 $v$
- upper-air mixing ratio $q$

Within each upper-air field, the data is organized as a series of $n_{\text{levai}}$ levels, with their respective pressure values in descending order.

The surface fields appear in the following order: sea-level pressure $p_s$, sea-level zonal wind $u_s$, sea-level meridional wind $v_s$.

- **V-vectors:** These multivariate forecast attribute vectors are calculated from G-vectors in the routine `mvPHx()`, and passed to and returned from the routine `opPF()`. These vectors are sorted in lexicographic order by region, variable, level, and profile $(\varphi, \lambda)$ (Figure 48).

The routine `getAinc()` performs the following steps:

1. Determination of the number $n_{\text{kerv}}$ of segments in the G-vector:
   
   (a) Initialization: $n_{\text{kerv}} = 0$
   (b) If `want_hght` = `.TRUE.` then $n_{\text{kerv}} = n_{\text{kerv}} + n_{\text{levai}}$
   (c) If `want_uwnd` = `.TRUE.` then $n_{\text{kerv}} = n_{\text{kerv}} + n_{\text{levai}}$
   (d) If `want_vwnd` = `.TRUE.` then $n_{\text{kerv}} = n_{\text{kerv}} + n_{\text{levai}}$
   (e) If `want_mixr` = `.TRUE.` then $n_{\text{kerv}} = n_{\text{kerv}} + n_{\text{levai}}$
   (f) If `want_slp` = `.TRUE.` then $n_{\text{kerv}} = n_{\text{kerv}} + 1$
   (g) If `want_slu` = `.TRUE.` then $n_{\text{kerv}} = n_{\text{kerv}} + 1$
   (h) If `want_slv` = `.TRUE.` then $n_{\text{kerv}} = n_{\text{kerv}} + 1$

2. Allocation of workspace variables:

   - INTEGER `krbeg`(maxreg): Region start index for QEA attribute vectors
   - INTEGER `krlen`(maxreg): Region lengths for QEA attribute vectors
   - INTEGER `kpmap`(mxkerv): Vertical level index for QEA attribute vectors.
   - INTEGER `ktmap`(mxkerv): 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 `rlat`(mxG): QEA grid latitude values.
   - REAL `rlon`(mxG): QEA grid longitude values.
   - REAL `rlev`(mxG): QEA grid vertical level values.
   - INTEGER `ktbeg`(ktmax, maxreg): Beginning points of region/datatype segments in QEA grid attribute vectors.
   - INTEGER `ktlen`(ktmax, maxreg): Lengths of region/datatype segments in QEA grid attribute vectors.
   - REAL `GsigF`(mxG, mxkerv): Forecast error standard deviations $\sigma_f$ 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 `kpmap(1:mxkerv)` and data type `ktmap(1:mxkerv)` 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 \texttt{gridxx}():

\[
\texttt{call gridxx ( iaidim, jaidim, nG,} \\
\texttt{krbegG, krlenG, invperm, rlatG, rlonG,} \\
\texttt{xgrid, ygrid, zgrid, ier)}
\]

The regular analysis grid is \( idaidim \) latitudes by \( jdaidim \) longitudes, and \( nG \) is the number of QEA gridpoints \( (nG < idaidim \times jaidim) \) 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 \texttt{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:

\[
\begin{align*}
ktbegG(1:ktmapG(kv),1:maxreg) & = 0 \\
ktlenG(1:ktmapG(kv)-1,1:maxreg) & = 0 \\
ktlenG(1:ktmapG(kv),1:maxreg) & = krlen(1:maxreg) \\
ktbegG(ktmapG(kv)+1,ktmax,1:maxreg) & = 0 \\
ktlenG(ktmapG(kv)+1,ktmax,1:maxreg) & = krlen(1:maxreg)
\end{align*}
\]

(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 \( \sigma_f \) for this \( kv \) slice are calculated by interpolating input values of \( \sigma_f \) 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 \texttt{intpsigF}:

\[
\texttt{call intpsigF(iaidim, jaidim, nlevai, plevai,} \\
\texttt{slpsigF, ussigF, vssigF, HHSigF, uusigF, vvsigF, qqsigF,} \\
\texttt{maxreg, krbegG, krlenG, kmax, ktbegG, ktlenG,} \\
\texttt{nG, rlatG, rlonG, rlevG, GsigF(1,kv))}
\]

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 \( P^T H^T \) and calculation \( P^T H^T x \) is accomplished by calling \texttt{mvPHx}():

\[
\begin{align*}
\texttt{call mvPHx ( nbamdex, maxreg, iregbeg, ireglen, ityplen,} & \\
\texttt{nnobs, nobs, rlats, rons, rlevs, sigF,} & \\
\texttt{maxreg, krbegG, krlenG, nG, rlatG, rlonG,} & \\
\texttt{nkvg, kmapG, kpmapG, nlevai, plevai,} & \\
\texttt{mxG, GsigF, i, nnobs, xvec,} & \\
\texttt{mxG, mxvkvG, Gout)}
\end{align*}
\]
The array $G_{\text{out}}(1:n_{\text{kvG}}*n_{G})$ contains the output AI on the QEA grid. A complete discussion of $\text{mvPHx}()$ 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 $\text{Gea211}()$:

$$\text{call Gea211( nG, n_{\text{kvG}}, \text{ktmapG}, \text{kpmapG}, 1, mxG, mx_{\text{kvG}}, G_{\text{out}}, \% iaidim, jaidim, nlevai, }$$

$$\text{HHai, uuai, vvai, qqai, slpai, usai, vsai \quad )}$$

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 $P^f H^T$ — $\text{mvPHx}()$

The global matrix multiplication $P^f H^T x$ is implemented in the routine $\text{mvPHx}()$:

$$\text{subroutine mvPHx( nbandmx, n_{krX}, kr_{locX}, kr_{lenX}, kt_{lenX}, \& n_X, rlatX, rlonX, rlevX, X_{sigF}, \& n_{krG}, kr_{locG}, kr_{lenG}, n_{G}, rlatG, rlonG, \& nk_{G}, \text{ktmapG}, \text{kpmapG}, nlevG, plevG, ld_{nG}, G_{sigF}, \& nvecs, ld_{X}, X_{vec}, ld_{nvkG}, G_{out} \quad )}$$

This routine takes various inputs organized as G-vectors on the QEA grid and observation vectors, calculates $w^a - w^f = P^f H^T x$, and returns the AI $(w^a - w^f)$ 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()`.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Verbosity Control Flag</td>
</tr>
<tr>
<td>lverb</td>
<td>INTEGER</td>
<td>IN</td>
<td>Diagnostic Output Device</td>
</tr>
<tr>
<td>nbndmx</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Matrix Bands</td>
</tr>
<tr>
<td>nnobs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Observations</td>
</tr>
<tr>
<td>iregbeg</td>
<td>INTEGER(maxreg)</td>
<td>IN</td>
<td>Region Start Index</td>
</tr>
<tr>
<td>ireglen</td>
<td>INTEGER(maxreg)</td>
<td>IN</td>
<td>Region Length</td>
</tr>
<tr>
<td>ityplen</td>
<td>INTEGER(ktmax,maxreg)</td>
<td>IN</td>
<td>Region/Datatype Segment Length</td>
</tr>
<tr>
<td>xvec</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Intermediate vector $\mathbf{x}$</td>
</tr>
<tr>
<td>rlat</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Latitude</td>
</tr>
<tr>
<td>rlon</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Longitude</td>
</tr>
<tr>
<td>rlev</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>Pressure (hPa)</td>
</tr>
<tr>
<td>sigF</td>
<td>REAL(nnobs)</td>
<td>IN</td>
<td>$\sigma_f$</td>
</tr>
<tr>
<td>iaidim</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number Analysis Latitudes</td>
</tr>
<tr>
<td>jaaidim</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number Analysis Longitudes</td>
</tr>
<tr>
<td>nlevai</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Analysis Levels</td>
</tr>
<tr>
<td>plevai</td>
<td>REAL(nlevai)</td>
<td>IN</td>
<td>Analysis Levels</td>
</tr>
<tr>
<td>usai</td>
<td>REAL(iaidim,jaidim)</td>
<td>OUT</td>
<td>Sea-level $u$ Al's</td>
</tr>
<tr>
<td>vsai</td>
<td>REAL(iaidim,jaidim)</td>
<td>OUT</td>
<td>Sea-level $v$ Al's</td>
</tr>
<tr>
<td>slpai</td>
<td>REAL(iaidim,jaidim)</td>
<td>OUT</td>
<td>Sea-level Pressure Al's</td>
</tr>
<tr>
<td>uuai</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>OUT</td>
<td>Upper-air $u$ Al's</td>
</tr>
<tr>
<td>vvai</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>OUT</td>
<td>Upper-air $v$ Al's</td>
</tr>
<tr>
<td>HHa</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>OUT</td>
<td>Upper-air $h$ Al's</td>
</tr>
<tr>
<td>qai</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>OUT</td>
<td>Upper-air $q$ Al's</td>
</tr>
<tr>
<td>ussigF</td>
<td>REAL(iaidim,jaidim)</td>
<td>IN</td>
<td>$\sigma_f$ for Sea-level $u$</td>
</tr>
<tr>
<td>vssigF</td>
<td>REAL(iaidim,jaidim)</td>
<td>IN</td>
<td>$\sigma_f$ for Sea-level $v$</td>
</tr>
<tr>
<td>slpsigF</td>
<td>REAL(iaidim,jaidim)</td>
<td>IN</td>
<td>$\sigma_f$ for Sea-level Pressure</td>
</tr>
<tr>
<td>uusigF</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>IN</td>
<td>$\sigma_f$ for Upper-air $u$</td>
</tr>
<tr>
<td>vvsigF</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>IN</td>
<td>$\sigma_f$ for Upper-air $v$</td>
</tr>
<tr>
<td>HHsigF</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>IN</td>
<td>$\sigma_f$ for Upper-air $h$</td>
</tr>
<tr>
<td>qqsigF</td>
<td>REAL(iaidim,jaidim,nlevai)</td>
<td>IN</td>
<td>$\sigma_f$ for Upper-air $q$</td>
</tr>
<tr>
<td>want slu</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Want $u_{sl}$ Al's</td>
</tr>
<tr>
<td>want slv</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Want $v_{sl}$ Al's</td>
</tr>
<tr>
<td>want slp</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Want Sea-level Pressure Al's</td>
</tr>
<tr>
<td>want uwind</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Want Upper-air $u$ Al's</td>
</tr>
<tr>
<td>want vwind</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Want Upper-air $v$ Al's</td>
</tr>
<tr>
<td>want hght</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Want Upper-air $h$ Al's</td>
</tr>
<tr>
<td>want mixr</td>
<td>LOGICAL</td>
<td>IN</td>
<td>Want Upper-air $q$ Al's</td>
</tr>
<tr>
<td>ierr</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Error Flag</td>
</tr>
</tbody>
</table>
input vector $x$ is sorted in lexicographic order, as illustrated in Figure 5. Block structure is imposed in $P^T H^T$ 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 $P^T H^T x$ 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.
   - $qrV_x(n.V), qrV_y(n.V), qrV_z(n.V), qmV_x(n.V), qmV_y(n.V), qmV_z(n.V), q1V_x(n.V), q1V_y(n.V)$: Cartesian components of $\hat{e}_r, \hat{e}_t, \hat{e}_m$ organized as V-vectors. The quantity $n.V$ is the total number of forecast values in the V-vector; $n.V = n_G * nkvG$.
   - $qrG_x(n.G), qrG_y(n.G), qrG_z(n.G), qmG_x(n.G), qmG_y(n.G), qmG_z(n.G), q1G_x(n.G), q1G_y(n.G)$: Cartesian components of $\hat{e}_r, \hat{e}_t, \hat{e}_m$ organized as G-vectors.
   - $ktablV(n.V)$ and $jtavbV(n.V)$: level and latitude indices for imat structures, organized as V-vectors.
   - $jtablG(n.G)$ and $vtablG(n.G)$: 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 $Xvec$.
   - $qrX_x(n.X), qrX_y(n.X), qrX_z(n.X), qmX_x(n.X), qmX_y(n.X), qmX_z(n.X), q1X_x(n.X), q1X_y(n.X)$: Cartesian components of $\hat{e}_r, \hat{e}_t, \hat{e}_m$ at observation locations.
   - $jtablX(n.X)$ and $vtablX(n.X)$: latitude index and interpolation weights for observation locations.
   - $ktablX(n.X)$ and $vtablX(n.X)$: 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 $P^T H^T x$ returned from op Pf().

3. Calculation of the Cartesian components of the unit vectors $\hat{e}_r, \hat{e}_t, \hat{e}_m$ for the horizontal QEA grid points are calculated by calling the routine 112qvec():

   ```
call 112qvec(n_G,rlatG,rlonG,qrG_x,qrG_y,qrG_z, &
   qmG_x,qmG_y,qmG_z,q1G_x,q1G_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_x`, `qrV_y`, `qrV_z`, `qmV_x`, `qmV_y`, `qmV_z`, `qlV_x`, `qlV_y`, `jtabv`, and `VsigtG`. 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 \( w_{lev} \). This value is converted to the appropriate INTEGER index \( k_{lev} \) as follows:

\[
\begin{align*}
    k_{lev} &= \text{min}(\text{int}(w_{lev}), n_{veclev} - 1) \\
    w_{lev} &= w_{lev} - k_{lev} \\
    k_{lev} &= k_{lev} + \text{int}(w_{lev})
\end{align*}
\]

and the appropriate values of \( k_{lev} \) are stored in the array \( k_{tabV} \).

8. Calculation of the observation grid attributes \( qr_{X,x}, qr_{X,y}, qr_{X,z}, qm_{X,x}, qm_{X,y}, qm_{X,z}, qL_{X,x}, qL_{X,y}, j_{tabX}, \) and \( k_{tabX} \). The Cartesian components of the unit vectors \( \hat{e}_r, \hat{e}_i, \) and \( \hat{e}_m \) are calculated using a call to \text{l12qvec()}:

\[
\text{call l12qvec(n_X, rlatX, rlonX, qr_{X,x}, qr_{X,y}, qr_{X,z}, & qm_{X,x}, qm_{X,y}, qm_{X,z}, qL_{X,x}, qL_{X,y})}
\]

9. The observation IMAT latitude table indices \( j_{tabX} \) are calculated by calling \text{slintab()}:

\[
\text{call slintab(roundoff, nveclat, veclats, n_X, rlatX, j_{tabX}, v_{tabX})}
\]

10. The observation IMAT level table indices \( k_{tabX} \) are calculated by calling \text{slogtab()}:

\[
\text{call slogtab(roundoff, nveclev, pveclev, n_X, rlevX, k_{tabX}, w_{tabX})}
\]

11. The computation \( P^j H^{T} x \) is performed in this forecast V-vector/Observation Vector representation by calling the routine \text{op_Pf()}:  

\[
\text{call op_Pf(n_{bandmx}, kind_{covF}.or.kind_{covS}.or.kind_{covV}, & n_{krG}, kr_{locV}, kr_{lenV}, k_{max}, kr_{locV}, kr_{lenV}, & n_{V}, k_{tabV}, j_{tabV}, v_{sigF}, & qr_{V,x}, qr_{V,y}, qr_{V,z}, qm_{V,x}, qm_{V,y}, qm_{V,z}, qL_{V,x}, qL_{V,y}, & n_{krX}, kr_{locX}, kr_{lenX}, k_{max}, kr_{locX}, kr_{lenX}, & n_{X}, k_{tabX}, j_{tabX}, x_{sigF}, & qr_{X,x}, qr_{X,y}, qr_{X,z}, qm_{X,x}, qm_{X,y}, qm_{X,z}, qL_{X,x}, qL_{X,y}, & n_{vecs}, X_{vec}, n_{V}, V_{out}(1:1))}
\]

The AI in V-vector form is returned in \( V_{out}(1:n_{V}, 1:n_{vecs}) \).

12. The V-vector AI \( V_{out} \) is then mapped to its G-vector representation \( G_{out} \):

\[
\begin{align*}
    \text{do } k_{m}=1, n_{krG} \\
    \text{lc}_{G} &= \text{kr}_{locG}(k_{r}) \\
    \text{ln}_{G} &= \text{kr}_{lenG}(k_{r}) \\
    \text{le}_{G} &= \text{lc}_{G} + \text{ln}_{G} - 1 \\
    \text{do } k_{v}=1, n_{kvG} \\
    \text{lc} &= \text{kr}_{locV}(k_{r}) + \text{kv}_{locV}(k_{v}, k_{r}) \\
    \text{ln} &= \text{kv}_{lenV}(k_{v}, k_{r}) \\
    \text{le} &= \text{lc} + \text{ln} - 1
\end{align*}
\]

\[
G_{out}(\text{lc}_{G}:\text{le}_{G}, k_{v}:1:n_{vecs}) = V_{out}(\text{lc}:+1:n_{vecs})
\]

end do
end do

13. The temporary arrays allocated in this routine are deallocated.

The routine \text{mvPHx()} returns the AI in G-vector form as \( G_{out}(\text{lc}_{G}:\text{le}_{G}, k_{v}:1:n_{vecs}) \).
Table 25: Arguments to the routine `mvPHx()`.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbandmx</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of bands in $P^J$</td>
</tr>
<tr>
<td>n.krX</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Observation Regions</td>
</tr>
<tr>
<td>kr_locX</td>
<td>INTEGER(n.krX)</td>
<td>IN</td>
<td>Region Start Index (Obs)</td>
</tr>
<tr>
<td>kr.lenX</td>
<td>INTEGER(n.krX)</td>
<td>IN</td>
<td>Region Length (Obs)</td>
</tr>
<tr>
<td>kt.lenX</td>
<td>INTEGER(ktmax,n.krX)</td>
<td>IN</td>
<td>Region/Datatype Segment Length (Obs)</td>
</tr>
<tr>
<td>n.X</td>
<td>INTEGER</td>
<td>IN</td>
<td>Intermediate vector x</td>
</tr>
<tr>
<td>rlatX</td>
<td>REAL(n.X)</td>
<td>IN</td>
<td>Latitude (Obs)</td>
</tr>
<tr>
<td>rlonX</td>
<td>REAL(n.X)</td>
<td>IN</td>
<td>Longitude (Obs)</td>
</tr>
<tr>
<td>rlevX</td>
<td>REAL(n.X)</td>
<td>IN</td>
<td>Pressure (hPa)</td>
</tr>
<tr>
<td>XsigF</td>
<td>REAL(n.X)</td>
<td>IN</td>
<td>$\sigma_f$ (obs space)</td>
</tr>
<tr>
<td>n.krG</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Forecast Regions</td>
</tr>
<tr>
<td>kr_locG</td>
<td>INTEGER(n.krG)</td>
<td>IN</td>
<td>Region Start Index (Fcast)</td>
</tr>
<tr>
<td>kr_lenG</td>
<td>INTEGER(n.krG)</td>
<td>IN</td>
<td>Region Length (Fcast)</td>
</tr>
<tr>
<td>kt_lenG</td>
<td>INTEGER(ktmax,n.krG)</td>
<td>IN</td>
<td>Region/Datatype Segment Length (Fcast)</td>
</tr>
<tr>
<td>n.G</td>
<td>INTEGER</td>
<td>IN</td>
<td># of horizontal QEA grid points</td>
</tr>
<tr>
<td>rlatG</td>
<td>REAL(n.G)</td>
<td>IN</td>
<td>Horizontal QEA latitudes</td>
</tr>
<tr>
<td>rlonG</td>
<td>REAL(n.G)</td>
<td>IN</td>
<td>Horizontal QEA longitudes</td>
</tr>
<tr>
<td>nkvg</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of kv-slices</td>
</tr>
<tr>
<td>ktmapG</td>
<td>INTEGER(nkvg)</td>
<td>IN</td>
<td>kt values for kv-slices</td>
</tr>
<tr>
<td>kmapG</td>
<td>INTEGER(nkvg)</td>
<td>IN</td>
<td>Pressure values for kv-slices</td>
</tr>
<tr>
<td>nlevG</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Analysis pressure levels</td>
</tr>
<tr>
<td>plevG</td>
<td>REAL(nlevG)</td>
<td>IN</td>
<td>Analysis pressure levels (hPa)</td>
</tr>
<tr>
<td>ldnGs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Length of GsigF</td>
</tr>
<tr>
<td>GsigF</td>
<td>REAL(ldnGs)</td>
<td>IN</td>
<td>G-vector forecast error</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of x vectors</td>
</tr>
<tr>
<td>ldnX</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of Xvec</td>
</tr>
<tr>
<td>Xvec</td>
<td>REAL(ldnX,nvecs)</td>
<td>IN</td>
<td>$x$</td>
</tr>
<tr>
<td>ldnG</td>
<td>INTEGER</td>
<td>IN</td>
<td>First leading dimension of Gout</td>
</tr>
<tr>
<td>ldnkvG</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of kv-slices in Gout</td>
</tr>
<tr>
<td>Gout</td>
<td>REAL(ldnG,ldnkvG,nvecs)</td>
<td>OUT</td>
<td>G-vector $P^JH^Tx$</td>
</tr>
</tbody>
</table>
Figure 48: Organization of the “V-vector” grid

9.2.1 V-vector Calculation of $P^T H^T x$ — op_Pf()

The calculation of $P^T H^T x$ 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, tabi, 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, jtabj, sigFj, &
    qrj_x, qrj_y, qrj_z, qmj_x, qmj_y, qmj_z, qlj_x, qlj_y, &
    nvecs, ldxj, xj, ldxj, Cxj, 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.:
The workspace used is dynamically allocated and includes:

- $\mathbf{T}_x(\text{ld}_x, n\text{vecs})$ and $\mathbf{T}_y(\text{ld}_y, n\text{vecs})$: intermediate result arrays.
- $k_t_i\_\text{len}_W(n\_k_t_i, n\_k_r_i)$ and $k_t_j\_\text{len}_W(n\_k_t_j, n\_k_r_j)$: block dimensions for the rows (V-vector) and columns (obs. vector).
- $\mathbf{w}_i(\text{ld}_i)$ and $\mathbf{w}_j(\text{ld}_j)$: Vectors used to store forecast $\psi$ and $\chi$ values.

The \texttt{INTEGER} parameter $\text{ld}_T$ is the maximum of $\text{ld}_x$ and $\text{ld}_y$.

The calculation of $P^j H^T \mathbf{x}$ is similar to the calculation of $(H P^j H^T + R)\mathbf{x}$ described in Section 8.3.

The first step in \texttt{covFFxpy()} is to initialize $\mathbf{T}_x(1:n_j,1:n\text{vecs})$ to zero. The term $\Gamma^h \Sigma^h \Gamma^h H^T \mathbf{x}$ is calculated using the following steps:

1. Initialize the temporary array $\mathbf{T}_x(1:n_x,1:n\text{vecs})$ to zero.
2. Calculate $\mathbf{T}_x = \Gamma^h H^T \mathbf{x}$ by calling the routine \texttt{aj Alf()}:
   
   ```
   call \texttt{aj Alf}(n\_k_r_j, k_r_j\_loc, k_r_j\_len, n\_k_t_j, k_t_j\_loc, k_t_j\_len, &
   n\_j, k_tab_j, j_tab_j, n\text{vecs}, ldx_j, x_j, ldT, T_x )
   ```
3. Calculate $\mathbf{T}_y = \Sigma^h \mathbf{T}_x$ by calling \texttt{mv diag()}:

   ```
   call \texttt{mv diag}(n\_k_r_j, k_r_j\_loc, k_r_j\_len, n\_k_t_j, k_t_j\_loc, k_t_j\_len, &
   n\_j, sigF_j, n\text{vecs}, ldT, T_x, ldT, T_y )
   ```
4. Set $\mathbf{T}_x(1:n_i,1:n\text{vecs}) = 0$. Calculate $\mathbf{T}_x = \mathbf{C}^h \mathbf{T}_y$ by calling the routine \texttt{rec\_Cxy()}:

   ```
   call \texttt{rec\_Cxy}(kind\_mat, kind\_covF, sparse,
   n\_k_r_i, k_r_i\_loc, k_r_i\_len, n\_k_t_i, k_t_i\_loc, k_t_i\_len, &
   n\_i, k_tab_i, &
   q_r_i\_x, q_r_i\_y, q_r_i\_z, q_m_i\_x, q_m_i\_y, q_m_i\_z, q_l_i\_x, q_l_i\_y, &
   n\_k_r_j, k_r_j\_loc, k_r_j\_len, n\_k_t_j, k_t_j\_loc, k_t_j\_len, &
   n\_j, k_tab_j, &
   q_r_j\_x, q_r_j\_y, q_r_j\_z, q_m_j\_x, q_m_j\_y, q_m_j\_z, q_l_j\_x, q_l_j\_y, &
   n\text{vecs}, ldT, T_y, ldT, T_x, istat )
   ```
5. Calculate $\mathbf{T}_y = \Sigma^h \mathbf{T}_x$ by calling the routine \texttt{mv diag()}:

   ```
   call \texttt{mv diag}(n\_k_r_i, k_r_i\_loc, k_r_i\_len, n\_k_t_i, k_t_i\_loc, k_t_i\_len, &
   n\_i, sigF_i, n\text{vecs}, ldT, T_x, ldT, T_y )
   ```
6. Set the work array $\mathbf{T}_x(1:n_i,1:n\text{vecs}) = 0$. Calculate $\mathbf{T}_x = \Gamma^h \mathbf{T}_y$ by calling the routine \texttt{mv Alf()}:

   ```
   call \texttt{mv Alf}(n\_k_r_i, k_r_i\_loc, k_r_i\_len, n\_k_t_i, k_t_i\_loc, k_t_i\_len, &
   n\_i, k_tab_i, j_tab_i, n\text{vecs}, ldT, T_y, n\_x, T_x )
   ```
The term $\Gamma^h \Sigma^h C^h \Sigma^h \Gamma^h T^T x$ is stored in $T(x(1:n_i,1:nvecs))$, and added to the result $y$ stored in $y(i(1:n_i,1:nvecs))$:

$$yi(1:n_i,1:nvecs) = yi(1:n_i,1:nvecs) + Tx(1:n_i,1:nvecs)$$

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 $x$ that contain variables that are either upper-air or surface wind components are stored in dynamically allocated arrays $k_tilenW(1:n_kti,1:n_kri)$ and $k_tj_lenW(l:nlrtj,1:n-krj)$. The forecast error standard deviations $\sigma^\psi$ and $\sigma^x$ are stored in $sigWi(1:n_i)$ and $sigWj(1:n_j)$. The windfield segment indexing information is set in $kti_lenWi$ and $ktj_lenWj$ as follows:

```c
kti_lenW(:, :) = 0
kti_lenW(ktus, :) = kti_len(ktus, :)
k_tilenW(ktvs, :) = kti_len(ktvs, :)
k_tilenW(ktuu, :) = kti_len(ktuu, :)
k_tilenW(ktvv, :) = kti_len(ktvv, :)
ktj_lenW(:, :) = 0
ktj_lenW(ktus, :) = ktj_len(ktus, :)
k_tj_lenW(ktvs, :) = ktj_len(ktvs, :)
k_tj_lenW(ktuu, :) = ktj_len(ktuu, :)
k_tj_lenW(ktvv, :) = ktj_len(ktvv, :)
```

The vector $\Gamma^\psi \Sigma^\psi C^\psi \Sigma^\psi \Gamma^\psi x$ 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 $\sigma^\psi$ obtained from the IMAT structure $FEsigS_imat$. This is implemented as a pair of calls to the routine `getivec()`.
   ```c
   call getivec(MXvelcelv, MXvelclat, FEsigS_imat, &
               n_kri, kri_loc, kri_len, n_kti, kti_loc, kti_lenW, &
               n_i, ktabi, jtabi, sigWi
   call getivec(MXvelcelv, MXvelclat, 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 $Tx = \Gamma^\psi T x$ by calling the routine `aj_Bet()`:
   ```c
   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()`:
   ```c
   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 \( \text{T}(1:n,1:nvecs) = 0 \). Calculate \( \text{T} = C^\psi \text{Ty} \). This is accomplished by calling the rectangular operator routine \text{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 \( \text{Ty} = \Sigma^\psi \text{Tx} \) by calling the routine \text{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 \( \text{T}(1:n,1:nvecs) = 0 \). Calculate \( \text{T} = \Gamma^\psi \text{Ty} \) by calling the routine \text{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 \( \Gamma^\psi \Sigma^\psi C^\psi \Sigma^\psi \Gamma^\psi T x \) is stored in \( \text{T}(1:n,1:nvecs) \), and added to the result \( y \) stored in \( y(1:n,1:nvecs) \):

\[
y(1:n,1:nvecs) = y(1:n,1:nvecs) + \text{T}(1:n,1:nvecs)
\]

The final error covariance operator term \( \Gamma^x \Sigma^x C^x \Sigma^x \Gamma^x T x \) is calculated for both the upper-air and surface analyses as follows:

1. Initialize the wind error streamfunction standard deviation temporary arrays \( \text{sigWi}(1:n,1) = 0 \) and \( \text{sigWj}(1:n,1) = 0 \).

2. Fill in the necessary elements of \( \text{sigWi} \) and \( \text{sigWj} \) with their respective values of \( \sigma^x \) obtained from the IMAT structure \text{FEsigS.imat} \). This is implemented as a pair of calls to the routine \text{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 \( \text{T}(1:n,1:nvecs) = 0 \).

4. Calculate \( \text{T} = \Gamma^x T x \) by calling the adjoint routine \text{aj_Gam()}:

   call aj_Gam(n_krj,krj_loc,krj_len,n_ktj,ktj_loc,ktj_lenW, &
   n_j,ktabj,jtabj,nvecs,ldT,Tx)

5. Calculate \( \text{Ty} = \Sigma^x \text{T} x \) by calling \text{mv.diag()}:
Figure 49: Calling tree for op_Pf().

call mv_diag(n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW, &
           n_i,sigWnvecs,ldT,Tx,ldT,Ty)

6. Set Tx(1:n_i,1:nvecs) = 0. Calculate Tx = C\nTy. 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_kri,kri_loc,kri_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 Ty = \Sigma Tx by calling the routine mv_diag():

call mv_diag(n_kri,kri_loc,kri_len,n_kti,kti_loc,kti_lenW, &
           n_i,sigWnvecs,ldT,Tx,ldT,Ty)

8. Set the work array Tx(1:n_i,1:nvecs) = 0. Calculate Tx = \Gamma \nTy 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 \Sigma C\n\Sigma \Gamma 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 \Pi H_T 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 \Pi H_T 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 \texttt{opPf()}. 

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind_mat</td>
<td>INTEGER</td>
<td>IN</td>
<td>Matrix Type</td>
</tr>
<tr>
<td>kind_cov</td>
<td>INTEGER</td>
<td>IN</td>
<td>Covariance Type</td>
</tr>
<tr>
<td>n_kri</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>kri_loc</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Index of Region Start</td>
</tr>
<tr>
<td>kri_len</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Region Lengths</td>
</tr>
<tr>
<td>n_kti</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Data Types</td>
</tr>
<tr>
<td>ktl_loc</td>
<td>INTEGER(n_kt,n_kr)</td>
<td>IN</td>
<td>Start of each \texttt{kt} block for each region</td>
</tr>
<tr>
<td>ktl_len</td>
<td>INTEGER(n_kt,n_kr)</td>
<td>IN</td>
<td>Length of each \texttt{kt} block for each region</td>
</tr>
<tr>
<td>i</td>
<td>INTEGER</td>
<td>IN</td>
<td>Dimension of Attribute Arrays</td>
</tr>
<tr>
<td>ktabi</td>
<td>INTEGER(n.x)</td>
<td>IN</td>
<td>Level Index for Look-Up Arrays</td>
</tr>
<tr>
<td>jtabi</td>
<td>INTEGER(n.x)</td>
<td>IN</td>
<td>Latitude Index for Look-Up Tables</td>
</tr>
<tr>
<td>sigFi</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$\sigma_f$</td>
</tr>
<tr>
<td>qri.x</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qri.y</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qri.z</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qmi.x</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>qmi.y</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>qmi.z</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>qli.x</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_i$</td>
</tr>
<tr>
<td>qli.y</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_i$</td>
</tr>
<tr>
<td>n_krj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>krj_loc</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Index of Region Start</td>
</tr>
<tr>
<td>krj_len</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Region Lengths</td>
</tr>
<tr>
<td>n_ktj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Data Types</td>
</tr>
<tr>
<td>ktj_loc</td>
<td>INTEGER(n_kt,n_kr)</td>
<td>IN</td>
<td>Start of each \texttt{kt} block for each region</td>
</tr>
<tr>
<td>ktj_len</td>
<td>INTEGER(n_kt,n_kr)</td>
<td>IN</td>
<td>Length of each \texttt{kt} block for each region</td>
</tr>
<tr>
<td>j</td>
<td>INTEGER</td>
<td>IN</td>
<td>Dimension of Attribute Arrays</td>
</tr>
<tr>
<td>ktabj</td>
<td>INTEGER(n.j)</td>
<td>IN</td>
<td>Level Index for Look-Up Tables</td>
</tr>
<tr>
<td>jtabj</td>
<td>INTEGER(n.j)</td>
<td>IN</td>
<td>Latitude Index for Look-Up Tables</td>
</tr>
<tr>
<td>sigFj</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$\sigma_f$</td>
</tr>
<tr>
<td>qrij.x</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qrij.y</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qrij.z</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>qmj.x</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>qmj.y</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>qmj.z</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>qlj.x</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_i$</td>
</tr>
<tr>
<td>qlj.y</td>
<td>REAL(n.j)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_i$</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of vectors</td>
</tr>
<tr>
<td>ldxj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $x$</td>
</tr>
<tr>
<td>xj</td>
<td>REAL(ldxj,nvecs)</td>
<td>IN</td>
<td>Input vectors $x$</td>
</tr>
<tr>
<td>ldCxj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $Cx$</td>
</tr>
<tr>
<td>Cxj</td>
<td>REAL(ldCxj,nvecs)</td>
<td>OUT</td>
<td>Output vectors $Mx$</td>
</tr>
</tbody>
</table>
10 Stage III: Software Implementation the "Lower-Level" Operators in PSAS

10.1 Mass-coupled Height/Wind Forecast Error Covariances $\Gamma^h$

The multivariate height/wind error covariance models used in the PSAS are described in [Guo et al., 1998] Sec. 4.

The multivariate operator $\Gamma^h$ and its transpose $\Gamma^hT$ are implemented in the routines mv.alf() and aj.alf(), respectively.

```fortran
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. $\text{nsize} > \text{ldH}$ and $\text{nsize} > \text{ldX}$.
   b. Surface wind variable blocks of unequal length, i.e., $\text{kt_len(ktus,kr)} \neq \text{kt_len(ktvs,kr)}$ for any region index $kr$.
   c. Upper-air wind variable blocks of unequal length, i.e., $\text{kt_len(ktuu,kr)} \neq \text{kt_len(ktvv,kr)}$ 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 $\alpha_{um_k}$ and $\alpha_{ul_k}$ are stored in the array $a.u(1:\text{nsize})$ and the coefficients $\alpha_{um_k}$ and $\alpha_{ul_k}$ are stored in the array $a.v(1:\text{nsize})$. The parameters $\alpha_{um_k}$, $\alpha_{ul_k}$, $\alpha_{um_k}$, and $\alpha_{ul_k}$ are defined analytically in [Guo et al., 1998] Sec. 4.
   b. An integer array $\text{list_row}(1:\text{n.kt*n.kr})$ contains indices of the starting points of distinct segments of the input array $Hvec$.

3. The index structure $\text{list_row}$ is filled in by looping over each $kr/kt$ combination, counting the number of distinct $kr/kt$ segments $\text{nrow}$, and filling the entry $\text{list_row}(1:\text{nrow})$ 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 $Xvec = \Gamma^hHvec$ is performed in the loop described below over the index $ir \in \{1..\text{nrow}\}$:
   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, $\Gamma^h$ is the identity operator. The following steps are taken to calculate $\Gamma^h Hvec$ for these kr/kt segments:

i. The beginning and end indices for this segment, lc and le, respectively, are determined:
   
   \[ \begin{align*}
   lc &= kr_{\text{loc}}(kr) + kt_{\text{loc}}(kt, kr) \\
   le &= lc + kt\_len(kt, kr) - 1.
   \end{align*} \]

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 $\Gamma^h Hvec$ 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:
   
   \[ \begin{align*}
   lc_u &= kr_{\text{loc}}(kr) + kt_{\text{loc}}(kt_u, kr) \\
   le_u &= lc_u + kt\_len(kt_u, kr) - 1 \\
   lc_v &= kr_{\text{loc}}(kr) + kt_{\text{loc}}(kt_v, kr) \\
   le_v &= lc_v + kt\_len(kt_v, kr) - 1.
   \end{align*} \]

ii. The mapping indices lc_u, le_u, lc_v, and le_v are also used to define indices to the IMAT tables Aumimat, Avmimat, Aulimat, and Avlimat:

   \[ \begin{align*}
   lc_m &= lc_u \\
   le_m &= le_u \\
   lc_l &= lc_v \\
   le_l &= le_v.
   \end{align*} \]

iii. The values of $\alpha_{um}$, $\alpha_{om}$, $\alpha_{ul}$, and $\alpha_{vl}$ are assigned from the IMAT tables Aumimat, Avmimat, Aulimat, and Avlimat respectively:

   \[ \begin{align*}
   a_u(lc_m:le_m) &= (Aumimat(ktab(i), jtab(i)), i=lc_u, le_u) \\
   a_v(lc_m:le_m) &= (Avmimat(ktab(i), jtab(i)), i=lc_v, le_v) \\
   a_u(lc_l:le_l) &= (Aulimat(ktab(i), jtab(i)), i=lc_u, le_t) \\
   a_v(lc_l:le_l) &= (Avlimat(ktab(i), jtab(i)), i=lc_v, le_v).
   \end{align*} \]

iv. Finally, the matrix-vector multiplication is applied to the input vector segment Hvec in the following loop:

   \[ \begin{align*}
   \text{do} & \text{ ivec}=1,nvecs \\
   Xvec & (lc_u:le_u, ivec) = Xvec(lc_m:le_m, ivec) + a_u(lc_m:le_m) Hvec(lc_m:le_m, ivec) \\
   & a_u(lc_l:le_l) Hvec(lc_l:le_l, ivec) \\
   Xvec & (lc_v:le_v, ivec) = Xvec(lc_m:le_m, ivec) + a_v(lc_m:le_m) Hvec(lc_m:le_m, ivec) \\
   & a_v(lc_l:le_l) Hvec(lc_l:le_l, ivec) \\
   \text{end do.}
   \end{align*} \]

The Transpose $\Gamma^h T$: The transpose $\Gamma^k T$ is implemented in the routine aj_Alf:
Table 27: Summary of arguments to \texttt{mvAlf()}.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{n.kr}</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>\texttt{kr.loc}</td>
<td>INTEGER(n.kr)</td>
<td>IN</td>
<td>Index of Region Start</td>
</tr>
<tr>
<td>\texttt{kr.len}</td>
<td>INTEGER(n.kr)</td>
<td>IN</td>
<td>Region Lengths</td>
</tr>
<tr>
<td>\texttt{n.kt}</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>\texttt{kt.loc}</td>
<td>INTEGER(n.kt,n.kr)</td>
<td>IN</td>
<td>Start of each \texttt{kt} block for each region</td>
</tr>
<tr>
<td>\texttt{kt.len}</td>
<td>INTEGER(n.kt,n.kr)</td>
<td>IN</td>
<td>Length of each \texttt{kt} block for each region</td>
</tr>
<tr>
<td>\texttt{nsize}</td>
<td>INTEGER</td>
<td>IN</td>
<td>Dimension of \texttt{Hvec} and \texttt{Xvec}</td>
</tr>
<tr>
<td>\texttt{ktab}</td>
<td>INTEGER(nsize)</td>
<td>IN</td>
<td>Level reference for lookup tables</td>
</tr>
<tr>
<td>\texttt{jtab}</td>
<td>INTEGER(nsize)</td>
<td>IN</td>
<td>Latitude reference for lookup tables</td>
</tr>
<tr>
<td>\texttt{nvecs}</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of input/output vectors</td>
</tr>
<tr>
<td>\texttt{ldH}</td>
<td>INTEGER</td>
<td>IN</td>
<td>Dimension of each input vector \texttt{h}</td>
</tr>
<tr>
<td>\texttt{Hvec}</td>
<td>REAL(1dH,nvecs)</td>
<td>IN</td>
<td>Input vectors \texttt{h}</td>
</tr>
<tr>
<td>\texttt{ldX}</td>
<td>INTEGER</td>
<td>IN</td>
<td>Dimension of each output vector \texttt{x}</td>
</tr>
<tr>
<td>\texttt{Xvec}</td>
<td>REAL(1dX,nvecs)</td>
<td>OUT</td>
<td>Output vectors \texttt{x}</td>
</tr>
</tbody>
</table>

The arguments for \texttt{aj.Alf} are as in Table 27, with the exception that intents of \texttt{Xvec} and \texttt{Hvec} are now \texttt{IN} and \texttt{OUT}, respectively. The only substantial difference between this routine and \texttt{mv.Alf} is the matrix-vector multiplication. In \texttt{aj.Alf}, the matrix-vector multiplication is implemented using the following loop:

```fortran
    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 $\Gamma^w$, and $\Gamma^x$

The mass-decoupled height/wind error covariance models used in the PSAS are described in [Guo et al., 1998] Sec. 4.

The global operator $\Gamma^w$ is implemented in the routine \texttt{mv.Bet}:

```fortran
    subroutine mv.Bet( n.kr,kr.loc,kr.len, &
        n.kt,kt.loc,kt.len, &
        nsize, ktab, jtab, &
        nvecs, ldH, Hvec, &
        ldX, Xvec )
```

122
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

\[
\hat{\beta} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}
\]

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`:

\[
\begin{align*}
lc_u &= kr\_loc(kr) + kt\_loc(kt_u, kr) \\
le_u &= lc_u + kt\_len(kt_u, kr) - 1 \\
lc_v &= kr\_loc(kr) + kt\_loc(kt_v, kr) \\
le_v &= lc_v + kt\_len(kt_v, kr) - 1.
\end{align*}
\]

Output storage map indices to `Xvec` are then calculated:

\[
\begin{align*}
lc_m &= lc_u \\
le_m &= le_u \\
lc_l &= lc_v \\
le_l &= le_v.
\end{align*}
\]

Finally, the matrix-vector multiplication is applied for each `u` and `v` segment in this region:

\[
\begin{align*}
Xvec(lc_u:le_u, 1:nvecs) &= -Hvec(lc_m:le_m, 1:nvecs) \\
Xvec(lc_v:le_v, 1:nvecs) &= Hvec(lc_l:le_l, 1:nvecs).
\end{align*}
\]

The implementation of the transpose operator \(\Gamma^T\) is the routine `aj.Bet()`:

```fortran
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 \(\Gamma^T\) acts only on `kt`-segments corresponding to wind variables. The matrix-vector multiply is implemented using:

\[
\begin{align*}
Hvec(lc_m:le_m, 1:nvecs) &= -Xvec(lc_u:le_u, 1:nvecs) \\
Hvec(lc_l:le_l, 1:nvecs) &= Xvec(lc_v:le_v, 1:nvecs).
\end{align*}
\]
The global operator $\Gamma^x$ is implemented in the routine \texttt{mv\_Gam}:

\begin{verbatim}
subroutine mv_Gam( n_kr, kr_loc, kr_len, &
n_kt, kt_loc, kt_len, &
nsize, ktab, jtab, &
nvecs, ldH, Hvec, &
ldx, Xvec )
\end{verbatim}

The argument list for \texttt{mv\_Gam()} is identical to that for \texttt{mv\_Mf} – see Table 27. The indexing procedure using a temporary array \texttt{irow(1:nsize)} identical to that for \texttt{mv\_Gam()}, resulting in an index table with \texttt{nrow} nonzero entries.

The matrix-vector multiplication is applied by looping over the segments of \texttt{Hvec} referenced by the array \texttt{irow(1:nrow)}, and acts only segments for which \texttt{kt} indicates a wind datatype.

For each wind segment, the operator

$$
\beta^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},
$$

is defined symbolically in [Guo et al., 1998] Sec. 4. is applied to the input vector \texttt{Hvec}. This is accomplished for each $(u,v)$ pair in a given region \texttt{kr} by first calculating input storage map indices \texttt{lc\_u} and \texttt{le\_u} for $u$ and \texttt{lc\_v} and \texttt{le\_v} for $v$:

- $\texttt{lc\_u} = \texttt{kr\_loc(kr)} + \texttt{kt\_loc(kt\_u,kr)}$
- $\texttt{le\_u} = \texttt{lc\_u} + \texttt{kt\_len(kt\_u,kr)} - 1$
- $\texttt{lc\_v} = \texttt{kr\_loc(kr)} + \texttt{kt\_loc(kt\_v,kr)}$
- $\texttt{le\_v} = \texttt{lc\_v} + \texttt{kt\_len(kt\_v,kr)} - 1$.

Output storage map indices to \texttt{Xvec} are then calculated:

- $\texttt{lc\_m} = \texttt{lc\_u}$
- $\texttt{le\_m} = \texttt{le\_u}$
- $\texttt{lc\_l} = \texttt{lc\_v}$
- $\texttt{le\_l} = \texttt{le\_v}$.

The transpose operator $\Gamma^{xT}$ is implemented in the routine \texttt{aj\_Gam}:

\begin{verbatim}
subroutine aj_Gam( n_kr, kr_loc, kr_len, &
n_kt, kt_loc, kt_len, &
nsize, ktab, jtab, &
nvecs, ldH, Xvec, &
ldx, Hvec )
\end{verbatim}
The arguments to this routine are as in Table 27 with the single difference that the argument \texttt{Xvec} has intent \texttt{IN} and \texttt{Yvec} has intent \texttt{OUT}.

This routine is also similar to \texttt{mv_set}. The indexing scheme used above is employed, and the operator acts only on \texttt{kt}-segments corresponding to wind variables. The matrix-vector multiplication is implemented using:

\begin{align*}
\texttt{Xvec}(l_{c.m}:1:le.m, i:1:nvcs) = \texttt{Xvec}(le.v:1:le_v, i:1:nvcs). \\
\texttt{Yvec}(le_1:1:le_1, i:1:nvcs) = \texttt{Yvec}(le_u:1:le_u, i:1:nvcs).
\end{align*}

\subsection{10.3 Forecast/Observation Error Standard Deviation Operator $\Sigma$}

In this section, we describe the forecast error standard-deviation operators $\Sigma^f$, $\Sigma^v$, $\Sigma^w$, and $\Sigma^e$, and the observation error standard deviation operators and $\Sigma^o$ and $\Sigma^e$. These operators are applied to the vector $x$ using the routine \texttt{mv_diag}:

\begin{verbatim}
subroutine mv_diag(n, kr, loc, len, n_kt, loc, len, nsq, sigm, nvcs, ldx, x, ldv, y

A summary of the arguments to \texttt{mv_diag()} is presented in Table 28. The diagonal operator $\Sigma$ is represented by the data structure \texttt{sigm(1:nsq)} which contains the \texttt{nsq} nonzero elements of $\Sigma$. The input vectors $x$, represented by \texttt{x(1:ldx, 1:nvcs)} are scanned region-by-region, where regions are keyed using \texttt{kr_loc(kr)}. Within each region, these vectors are scanned type-by-type, where types are keyed using \texttt{kr_loc(kr, kr)}, to determine a \texttt{kr/kt} block of $x$ whose initial index is $le = kr_{loc}(kr) + kr_{loc}(kr)$ and final index is $le = le + k_t_{len}(kr, kr) - 1$. The operator $\Sigma$ is applied to each vector segment in this block using:

\begin{verbatim}
do ivec=1,nvcs
  y(lc:le, ivec) = sigm(lc:le)*x(lc:le, ivec)
end do.
\end{verbatim}

\subsection{10.4 The Symmetric Error Correlation Operator $= \texttt{sym_Cxpy}()$}

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 \texttt{sym_Cxpy()} constructs the matrices necessary to transform $x$ into $Mx$, where $M = HP^F K^E + R$ in the so-called \textit{factored operator formulation} described in [Guo et al., 1998].

These matrices are formed, and the matrix-vector product $Mx$ is computed in the routine \texttt{sym_Cxpy()}. The flow control diagram of \texttt{sym_Cxpy()} is illustrated in Figure 51.

The interface to \texttt{sym_Cxpy()} is:

\begin{verbatim}
subroutine sym_Cxpy(kind=mat, kind=gov, sparse, n, kr, loc, len, n_kt, loc, len, n, x, x, ks, ktab, q, r, q, r, q, m, x, m, y, q, m, z, q, l, x, q, l, y, nvcs, ldx, x, ldy, y, istat
\end{verbatim}
Table 28: Summary of arguments to mv.diag().

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n.kr</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>kr.loc</td>
<td>INTEGER(n.kr)</td>
<td>IN</td>
<td>Index of Region Start</td>
</tr>
<tr>
<td>kr.len</td>
<td>INTEGER(n.kr)</td>
<td>IN</td>
<td>Region Lengths</td>
</tr>
<tr>
<td>n.kt</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Data Types</td>
</tr>
<tr>
<td>kt.loc</td>
<td>INTEGER(n.kt,n.kr)</td>
<td>IN</td>
<td>Start of each kt block for each region</td>
</tr>
<tr>
<td>kt.len</td>
<td>INTEGER(n.kt,n.kr)</td>
<td>IN</td>
<td>Length of each kt block for each region</td>
</tr>
<tr>
<td>nsizex</td>
<td>INTEGER</td>
<td>IN</td>
<td>Dimension of Σ</td>
</tr>
<tr>
<td>sigm</td>
<td>REAL(nsize)</td>
<td>IN</td>
<td>Elements of Σ</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of vectors x</td>
</tr>
<tr>
<td>ldxy</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of elements in each output vector y</td>
</tr>
<tr>
<td>x</td>
<td>REAL(ldx,nvecs)</td>
<td>IN</td>
<td>Set of input vectors x</td>
</tr>
<tr>
<td>y</td>
<td>REAL(ldy,nvecs)</td>
<td>OUT</td>
<td>Set of output vectors y</td>
</tr>
</tbody>
</table>

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 ≤ n.x.
   (b) The leading dimension of the output vector l.dy ≤ n.x.
   (c) The matrix type flag kind.mat ∈ {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 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 $mrow = n_{kr} \times n_{kt}$. 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:

$$\text{list\_err}(1:nrow) = 0 .$$

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:

$$kr = (irow-1)/n_{kt} + 1$$
$$kt = \text{mod}(irow-1,n_{kt}) + 1 .$$

Next, the starting index $lc$ and ending index $le$ are calculated for this block:

$$lc = kr\_loc(kr) + kt\_loc(kt,kr)$$
The actual calculation of this block of the operator and its application to \( x(1c:le,1:nvecs) \) is a call to the routine \( Cprodl() \) described in Section 10.4.1 below:

```plaintext
call Cprodl(kind_cov, kr, kt, 
        ln, kx(1c), ks(1c), ktab(1c), & 
        qr_x(1c), qr_y(1c), qr_z(1c), & 
        qm_x(1c), qm_y(1c), qm_z(1c), & 
        q1_x(1c), q1_y(1c), & 
        nvecs, ldv, x(1c,1), n_x, Cx(1c,1), 
        ierr).
```

The result of the matrix-vector multiplication is returned in the vector \( Cx(1c:le,1:nvecs) \), and the status flag \( ierr \) for this block is stored in \( list_{err}(irow) \). If \( ierr = 0 \), then \( Cprodl() \) returned successfully, and the partial product \( Cx(1c:le,1:nvecs) \) is added to the output vector \( y \):

\[
y(1c:le,1:nvecs) = Cx(1c:le,1:nvecs) + y(1c:le,1:nvecs).
\]

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 \( \text{kind}_\text{mat} = \text{kind}_\text{Umat} \), to indicate that there are no horizontal error correlations. In this case only the diagonal \( kr/kt \) band is computed, and \( Cprodl() \) returns.

**Off-diagonal Bands:** The action of the off-diagonal bands is a loop over the index \( j_{\text{band}} = 1, m_{\text{band}}-1 \), where \( m_{\text{band}} \) is the number of bands in the operator.

There are two loops internal to this loop over \( j \). The first internal loop over \( i_{\text{row}} = 1, m_{\text{row}} \) indexes the so-called nonzero blocks. The number of nonzero block is set to \( n_{\text{row}} \) in this loop. The second internal loop over \( i_{\text{r}} = 1, n_{\text{row}} \) 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 \( \text{kind}_\text{mat} = \text{kind}_\text{Rmat} \), to indicate that the error correlation model is currently univariate.
and the number of nonzero bands is \( m_{\text{band}} = n_{\text{kt}} \). For \( \text{kind}_{\text{mat}} \neq \text{kind}_{\text{Rmat}} \), the number of nonzero bands is set to \( m_{\text{band}} = n_{\text{kr}} * n_{\text{kt}} \).

For each band, the maximum number of row segments \( m_{\text{row}} \) is set, and the nonzero operator block counter \( n_{\text{row}} \) is initialized:

\[
\begin{align*}
  m_{\text{row}} &= n_{\text{kt}} * n_{\text{kr}} - j_{\text{band}} \\
n_{\text{row}} &= 0.
\end{align*}
\]

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 \( i_{\text{row}} = 1, m_{\text{row}} \). The local column index \( j_{\text{col}} \) is defined and the dimensions of each block are calculated:

\[
\begin{align*}
  j_{\text{col}} &= i_{\text{row}} + j_{\text{band}}.
\end{align*}
\]

Row indices/dimensions:

\[
\begin{align*}
  kr_{-i} &= (i_{\text{row}} - 1) / n_{\text{kt}} + 1 \\
  kt_{-i} &= \text{mod}(i_{\text{row}} - 1, n_{\text{kt}}) + 1 \\
  ln_{-i} &= \text{kt}_n(kt_{-i}, kr_{-i}).
\end{align*}
\]

Column indices/dimensions:

\[
\begin{align*}
  kr_{-j} &= (j_{\text{col}} - 1) / n_{\text{kt}} + 1 \\
  kt_{-j} &= \text{mod}(j_{\text{col}} - 1, n_{\text{kt}}) + 1 \\
  ln_{-j} &= \text{kt}_n(kt_{-j}, kr_{-j}).
\end{align*}
\]

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 \( \text{sparse(kind}_{\text{mat}}, \text{kind}_{\text{cov}}, kr_{-i}, kt_{-i}, kr_{-j}, kt_{-j}) \) is false.

For each nonzero block, the total number of blocks \( n_{\text{row}} \) is incremented, and the current row location \( i_{\text{row}} \) is stored in \( \text{list}_{\text{row}}(n_{\text{row}}) \).

At the end of the sweep through this band, the value of \( n_{\text{row}} \) is checked. If \( n_{\text{row}} = 0 \), 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 \( \text{list}_{\text{err}} \) and partial product arrays \( C_{xi} \) and \( C_{xj} \) are initialized:

\[
\begin{align*}
  \text{list}_{\text{err}}(1:n_{\text{row}}) &= 0 \\
  C_{xi}(1:n_{\text{row}}) &= 0.0 \\
  C_{xj}(1:n_{\text{row}}) &= 0.0 .
\end{align*}
\]
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:

\[
irow = \text{list\_row}(ir) \\
jcol = irow + jband.
\]

For each nonzero block, the region and datatype row indices \( kr\_i \) and \( kt\_i \) are calculated, along with row pointer \( l\_c\_i \) and block row dimension \( l\_n\_i \):

\[
kr\_i = \left( (irow-1)/n_{kt} + 1 \right) \\
kt\_i = \text{mod}(irow-1, n_{kt}) + 1 \\
l\_c\_i = kr\_loc(kr\_i) + kt\_loc(kt\_i, kr\_i) \\
l\_n\_i = kt\_len(kt\_i, kr\_i).
\]

Region and datatype column indices \( kr\_j \) and \( kt\_j \) are calculated, along with column pointer \( l\_c\_j \) and block column dimension \( l\_n\_j \):

\[
kr\_j = \left( (jcol-1)/n_{kt} + 1 \right) \\
kt\_j = \text{mod}(jcol-1, n_{kt}) + 1 \\
l\_c\_j = kr\_loc(kr\_j) + kt\_loc(kt\_j, kr\_j) \\
l\_n\_j = kt\_len(kt\_j, kr\_j).
\]

Once the location and dimensions of this operator block are determined, the operator is calculated and applied using the routine \( \text{Cprodx}() \) described in Section 10.4.2 below:

\[
\text{call Cprodx(kind\_cov,} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&} \\
\text{\&}
\]

The value of the error flag \( ier \) is stored in \( \text{list\_err}(ir) \). Once the loop over the blocks in this band is complete, the status array \( \text{list\_err}(1:nrow) \) is scanned for nonzero elements. If any are found, \( \text{sym\_Cxpy()} \) 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 \( C\_i \) and \( C\_j \):

\[
y(1:n\_x, 1:n\_vecs) = y(1:n\_x, 1:n\_vecs) + C\_i(1:n\_x, 1:n\_vecs) \\
\quad + C\_j(1:n\_x, 1:n\_vecs).
\]
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 Cprodl()

The calculation and application of the individual blocks in the diagonal band of symmetric operators is implemented in the routine Cprodl():

\[
\text{subroutine Cprodl(kind Cov, krt, knt, \textbf{kr}, \textbf{kx}, \textbf{ks}, \textbf{kt}, \textbf{qr}_x, \textbf{qr}_y, \textbf{qr}_z, \textbf{qm}_x, \textbf{qm}_y, \textbf{qm}_z, \textbf{ql}_x, \textbf{ql}_y, \textbf{nvecs}, ldX, x, ldCx, Cx, ierr)}
\]

The arguments to Cprodl() are described in Table 31.

The following conditions are tested in the initial step in Cprodl():

- 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_covU.

If any of above conditions is violated, Cprodl() returns with ierr = -1. If nvecs = 0 or ln = 0, no action is necessary and Cprodl() returns with ierr = 0.

The operator blocks for the diagonal band are symmetric, and thus the \(ln \times ln\) block is represented in upper triangular form in a dynamically allocated array \(\text{Corr}(ln*(ln+1)/2)\). 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 \(\text{Corr}(ln*(ln+1)/2)\).

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:

\[
do \text{ivec=1,nvecs}
\text{call SSPMV('U',ln,1..Corr,x(1,ivec),1,0.,Cx(1,ivec),1)}
end do.
\]

The temporary structure \(\text{Corr}\) is deallocated and Cprodl() returns.
Table 29: Summary of arguments to \texttt{sym.Cxpy()}.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind_mat</td>
<td>INTEGER</td>
<td>IN</td>
<td>Matrix Type</td>
</tr>
<tr>
<td>kind_c cov</td>
<td>INTEGER</td>
<td>IN</td>
<td>Covariance Type</td>
</tr>
<tr>
<td>sparse</td>
<td>INTEGER</td>
<td>IN</td>
<td>Matrix Type</td>
</tr>
<tr>
<td>n.kr</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions</td>
</tr>
<tr>
<td>kr.loc</td>
<td>INTEGER(n.kr)</td>
<td>IN</td>
<td>Index of Region Start</td>
</tr>
<tr>
<td>kr.len</td>
<td>INTEGER(n.kr)</td>
<td>IN</td>
<td>Region Lengths</td>
</tr>
<tr>
<td>n.kt</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Data Types</td>
</tr>
<tr>
<td>ke_loc</td>
<td>INTEGER(n.kt,n.kr)</td>
<td>IN</td>
<td>Start of each ke block for each region</td>
</tr>
<tr>
<td>ke.len</td>
<td>INTEGER(n.kt,n.kr)</td>
<td>IN</td>
<td>Length of each ke block for each region</td>
</tr>
<tr>
<td>n.x</td>
<td>INTEGER</td>
<td>IN</td>
<td>Dimension of Attribute Arrays</td>
</tr>
<tr>
<td>kx</td>
<td>INTEGER(n.x)</td>
<td>IN</td>
<td>Data Source Index</td>
</tr>
<tr>
<td>ks</td>
<td>INTEGER(n.x)</td>
<td>IN</td>
<td>Sounding Index</td>
</tr>
<tr>
<td>ktab</td>
<td>INTEGER(n.x)</td>
<td>IN</td>
<td>Level Index for Look-Up Tables</td>
</tr>
<tr>
<td>q.r.x</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>q.r.y</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>q.r.z</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$</td>
</tr>
<tr>
<td>q.m.x</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>q.m.y</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>q.m.z</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$</td>
</tr>
<tr>
<td>q.l.x</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_l$</td>
</tr>
<tr>
<td>q.l.y</td>
<td>REAL(n.x)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_l$</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of vectors</td>
</tr>
<tr>
<td>ldx</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $x$</td>
</tr>
<tr>
<td>x</td>
<td>REAL(ldx,nvecs)</td>
<td>IN</td>
<td>Input vectors $x$</td>
</tr>
<tr>
<td>ldy</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $y$</td>
</tr>
<tr>
<td>y</td>
<td>REAL(ldy,nvecs)</td>
<td>OUT</td>
<td>Output vectors $y$</td>
</tr>
<tr>
<td>istat</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Return Status</td>
</tr>
</tbody>
</table>

Table 30: Error Correlation Calculation Calls from \texttt{Cprod1()}.

<table>
<thead>
<tr>
<th>Value of kind.cov</th>
<th>Routine Called</th>
<th>Error Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind.covF</td>
<td>diagcorF()</td>
<td>Forecast $h$, Mass-coupled $(u, v)$, $q$</td>
</tr>
<tr>
<td>kind.covS</td>
<td>diagcorF()</td>
<td>Forecast Mass-decoupled Wind $(\psi)$</td>
</tr>
<tr>
<td>kind.covV</td>
<td>diagcorF()</td>
<td>Forecast Mass-decoupled Wind $(\chi)$</td>
</tr>
<tr>
<td>kind.covC</td>
<td>diagcorU()</td>
<td>Horizontally Correlated Obs.</td>
</tr>
<tr>
<td>kind.covU</td>
<td>diagcorU()</td>
<td>Horizontally Uncorrelated Obs.</td>
</tr>
</tbody>
</table>
Table 31: Summary of arguments to Cprodl().

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind_cov</td>
<td>INTEGER</td>
<td>IN</td>
<td>Covariance Type</td>
</tr>
<tr>
<td>kr</td>
<td>INTEGER</td>
<td>IN</td>
<td>Region Index</td>
</tr>
<tr>
<td>kt</td>
<td>INTEGER</td>
<td>IN</td>
<td>Datatype Index</td>
</tr>
<tr>
<td>ln</td>
<td>INTEGER</td>
<td>IN</td>
<td>Block Dimension</td>
</tr>
<tr>
<td>kx</td>
<td>INTEGER(ln)</td>
<td>IN</td>
<td>Data Source Index (observations)</td>
</tr>
<tr>
<td>ks</td>
<td>INTEGER(ln)</td>
<td>IN</td>
<td>Sounding Index (observations)</td>
</tr>
<tr>
<td>ktab</td>
<td>INTEGER(ln)</td>
<td>IN</td>
<td>Look-up Table Level Indices</td>
</tr>
<tr>
<td>qr.x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>x-component of ( \hat{e}_r )</td>
</tr>
<tr>
<td>qr.y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>y-component of ( \hat{e}_r )</td>
</tr>
<tr>
<td>qr.z</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>z-component of ( \hat{e}_r )</td>
</tr>
<tr>
<td>qm.x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>x-component of ( \hat{e}_m )</td>
</tr>
<tr>
<td>qm.y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>y-component of ( \hat{e}_m )</td>
</tr>
<tr>
<td>qm.z</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>z-component of ( \hat{e}_m )</td>
</tr>
<tr>
<td>q1.x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>x-component of ( \hat{e}_1 )</td>
</tr>
<tr>
<td>q1.y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>y-component of ( \hat{e}_1 )</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of vectors</td>
</tr>
<tr>
<td>ldX</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of ( \mathbf{x} )</td>
</tr>
<tr>
<td>x</td>
<td>REAL(ldX,nvecs)</td>
<td>IN</td>
<td>Input vectors ( \mathbf{x} )</td>
</tr>
<tr>
<td>ldCx</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of ( \mathbf{C} \mathbf{x} )</td>
</tr>
<tr>
<td>Cx</td>
<td>REAL(ldCx,nvecs)</td>
<td>OUT</td>
<td>Output vectors ( \mathbf{C} \mathbf{x} )</td>
</tr>
<tr>
<td>ierr</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Return Status</td>
</tr>
</tbody>
</table>
10.4.2 The Off-diagonal Block Operator \texttt{Cprodx()} 

The calculation and application of the individual blocks in the off-diagonal bands of symmetric operators is implemented in the routine \texttt{Cprodx()}:

\begin{verbatim}
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,hen_j,kxj,ktabj, &
    qrj_x,qrj_y,qrj_z, &
    qmj_x,qmj_y,qmj_z, &
    qlj_x,qlj_y, &
    nvecs,ldXi,xi,ldCxj,Cxj, &
    ldXj,xj,ldCxi,Cxi, &
    ierr )
\end{verbatim}

The arguments to \texttt{Cprodx()} are described in Table 33.

The following conditions on input data are tested in the initial step in \texttt{Cprodx()}:

- Input vector \texttt{i-segment length ldXi} \texttt{\geq leni}.
- Output vector \texttt{j-segment length ldCxj} \texttt{\geq leni}.
- Input vector \texttt{j-segment length ldXj} \texttt{\geq lenj}.
- Output vector \texttt{i-segment length ldCxi} \texttt{\geq lenj}.
- The type of error correlation block requested corresponding to the variable \texttt{kind_cov} is one of the following: \texttt{kind_covP, kind_covS, kind_covV, or kind_covC}.

If any of the above conditions is violated, \texttt{Cprodx()} returns with error flag \texttt{ierr} = -1.

There are some special execution cases that require at most direct assignment of output data:

- If \texttt{leni} and \texttt{lenj} are both zero, no action is necessary; \texttt{Cprodx()} returns.
- If \texttt{leni} = 0 and \texttt{lenj} > 0, set the output vector segments \texttt{Cxi(1:lenj,1:nvecs)} and return.
- If \texttt{lenj} = 0 and \texttt{leni} > 0, set the output vector segments \texttt{Cxj(1:leni,1:nvecs)} and return.

The error correlation block to be computed is stored in the dynamically allocated array \texttt{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 \texttt{kind_cov}. The routines called and their types are summarized in Table 32.

The error correlation operator values are returned in \texttt{Corr}, along with the matrix type stored in a CHARACTER variable \texttt{Mtyp}. There are three possible classes of values for \texttt{Mtyp}: 'T' or 't' for a transpose block, 'N' or 'n' for a non-transpose block, and 'Z' for a zero block.
The application of the block operator and its transpose to each of the \( \text{nvec} \) input vectors proceeds in a loop over the index \( \text{ivec} \), with the matrix-vector multiplication performed by a call to the BLAS function \( \text{SGEMV()} \).

Suppose that the operator block stored in \( \text{Corr} \) is a transposed block. The following pair of calls to \( \text{SGEMV()} \) perform matrix-vector segment multiplications of an upper triangular block within \( \text{Corr} \) and of its transpose residing in the lower triangular block of \( \text{Corr} \):

\[
\begin{align*}
call \text{SGEMV}('T', \text{lenj}, \text{leni}, 1..\text{Corr}, \text{lenj}, x_j(1, \text{ivec}), 1, \& 0., \text{Cxi}(1, \text{ivec}), 1) \\
call \text{SGEMV}('N', \text{lenj}, \text{leni}, 1..\text{Corr}, \text{lenj}, x_i(1, \text{ivec}), 1, \& 0., \text{Cxj}(1, \text{ivec}), 1)
\end{align*}
\]

Suppose that the operator block stored in \( \text{Corr} \) is a non-transposed block. The following pair of calls to \( \text{SGEMV()} \) perform matrix-vector segment multiplications of an upper triangular block within \( \text{Corr} \) and of its transpose residing in the lower triangular block of \( \text{Corr} \):

\[
\begin{align*}
call \text{SGEMV}('N', \text{leni}, \text{lenj}, 1..\text{Corr}, \text{leni}, x_j(1, \text{ivec}), 1, \& 0., \text{Cxi}(1, \text{ivec}), 1) \\
call \text{SGEMV}('T', \text{leni}, \text{lenj}, 1..\text{Corr}, \text{leni}, x_i(1, \text{ivec}), 1, \& 0., \text{Cxj}(1, \text{ivec}), 1)
\end{align*}
\]

For zero blocks, no multiplication is necessary:

\[
\begin{align*}
\text{Cxi(1: lenj, ivec)} &= 0. \\
\text{Cxj(1: leni, ivec)} &= 0.
\end{align*}
\]

Upon completion of the multiplications for each vector, \( \text{Corr} \) is deallocated and \( \text{Cprodx()} \) returns with \( \text{ierr} = 0 \).

### 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 \( \text{rec_Cxpy} \):
Table 33: Summary of arguments to Cprodx().

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind_cov</td>
<td>INTEGER</td>
<td>IN</td>
<td>Covariance Type</td>
</tr>
<tr>
<td>kri</td>
<td>INTEGER</td>
<td>IN</td>
<td>Region Index (rows)</td>
</tr>
<tr>
<td>kti</td>
<td>INTEGER</td>
<td>IN</td>
<td>Datatype Index (rows)</td>
</tr>
<tr>
<td>leni</td>
<td>INTEGER</td>
<td>IN</td>
<td>Block Row Dimension</td>
</tr>
<tr>
<td>kxi</td>
<td>INTEGER</td>
<td>IN</td>
<td>Data Source Index (rows)</td>
</tr>
<tr>
<td>ktabi</td>
<td>INTEGER(ln)</td>
<td>IN</td>
<td>Look-up Table Level Indices (rows)</td>
</tr>
<tr>
<td>qri_x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td>qri_y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td>qri_z</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td>qmi_x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td>qmi_y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td>qmi_z</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td>qli_x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_l$ (rows)</td>
</tr>
<tr>
<td>qli_y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_l$ (rows)</td>
</tr>
<tr>
<td>krti</td>
<td>INTEGER</td>
<td>IN</td>
<td>Region Index (columns)</td>
</tr>
<tr>
<td>kti</td>
<td>INTEGER</td>
<td>IN</td>
<td>Datatype Index (columns)</td>
</tr>
<tr>
<td>lenj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Block Row Dimension</td>
</tr>
<tr>
<td>kxj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Data Source Index (columns)</td>
</tr>
<tr>
<td>ktabj</td>
<td>INTEGER(ln)</td>
<td>IN</td>
<td>Look-up Table Level Indices (columns)</td>
</tr>
<tr>
<td>qrj_x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$ (columns)</td>
</tr>
<tr>
<td>qrj_y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$ (columns)</td>
</tr>
<tr>
<td>qrj_z</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$ (columns)</td>
</tr>
<tr>
<td>qmj_x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$ (columns)</td>
</tr>
<tr>
<td>qmj_y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$ (columns)</td>
</tr>
<tr>
<td>qmj_z</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$ (columns)</td>
</tr>
<tr>
<td>qlj_x</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_l$ (columns)</td>
</tr>
<tr>
<td>qlj_y</td>
<td>REAL(ln)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_l$ (columns)</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of vectors</td>
</tr>
<tr>
<td>ldXi</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $\mathbf{x}$</td>
</tr>
<tr>
<td>xi</td>
<td>REAL(ldX,nvecs)</td>
<td>IN</td>
<td>Input vectors for block $ji$</td>
</tr>
<tr>
<td>ldCxi</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $\mathbf{C}_{Xi}$</td>
</tr>
<tr>
<td>Cxi</td>
<td>REAL(ldCxi,nvecs)</td>
<td>OUT</td>
<td>Output vectors $C_{ij}\mathbf{x}$</td>
</tr>
<tr>
<td>ldXj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $\mathbf{x}$</td>
</tr>
<tr>
<td>xj</td>
<td>REAL(ldX,nvecs)</td>
<td>IN</td>
<td>Input vectors for block $ij$</td>
</tr>
<tr>
<td>ldCxj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $\mathbf{C}\mathbf{x}$</td>
</tr>
<tr>
<td>Cxj</td>
<td>REAL(ldCx,nvecs)</td>
<td>OUT</td>
<td>Output vector segment $C_{ij}\mathbf{x}$</td>
</tr>
<tr>
<td>ierr</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Return Status</td>
</tr>
</tbody>
</table>
The arguments to \texttt{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: \( n_i \leq ldy_i \).
- Input vector attribute lengths and vector lengths are compatible: \( n_j \leq ldx_j \).
- The requested operator type \( \texttt{kind.mat} \in \{ \texttt{kind.Umat, kind.Rmat, kind.3mat, kind.4mat, kind.5mat} \} \).

If any of the above conditions are violated, \texttt{rec.Cxpy()} returns with \( \texttt{istat} = -1 \).

Temporary workspace is allocated:

- \texttt{list.row(n.ktj*n.krj)}, an INTEGER list of operator blocks.
- \texttt{list.err(n.ktj*n.krj)}, an INTEGER success status for calculation/application of individual operator blocks.
- \texttt{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.
- \texttt{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 \( \texttt{nrow} \) 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:

\[
\begin{align*}
kri &= (irow-1)/n_kti + 1 \\
kti &= \text{mod}(irow-1, n_kti) + 1 \\
lni &= \text{kt} \_ \text{leni}(kti,kri).
\end{align*}
\]

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:

\[
\begin{align*}
krj &= (jcol-1)/n_ktj + 1 \\
ktj &= \text{mod}(jcol-1, n_ktj) + 1 \\
lnj &= \text{kt} \_ \text{lenj}(ktj,krj).
\end{align*}
\]

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:

\[
correlated(irow,jcol) = \text{not.} \ \text{sparse(kind_mat,kind_cov, kri,kti, krj,ktj}).
\]

Once the scan over $jcol$ is complete, a second scan over $jcol$ is performed to build the list of nonzero blocks:

\[
\begin{align*}
do \ jcol=1,n_ktj*n_krj \\
& \quad \text{if} (\text{correlated}(irow,jcol)) \text{ then} \\
& \quad \quad nrow=nrow+1 \\
& \quad \quad \text{list} \_ \text{row}(nrow)=irow \\
& \quad \quad \text{exit} \\
& \quad \text{endif} \\
& \end do.
\end{align*}
\]

This completes the row and column scan. The result is a set of row segment pointer indices $\text{list} \_ \text{row}(1:nrow)$. A final preparation step is to initialize the error flag array $\text{list} \_ \text{err}(1:nrow) = 0$.

**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 $irow = \text{list} \_ \text{row}(ir)$.
- the row region index $kri = (irow-1)/n_kti + 1$.
- the row datatype index $kti = \text{mod}(irow-1, n_kti) + 1$.
- the starting index in row of the block $lci = \text{kr} \_ \text{loci}(kri) + \text{kt} \_ \text{loci}(kti,kri)$.
- the extent in rows of the block $lni = \text{kt} \_ \text{leni}(kti,kri)$.
- the ending index in rows of the block $lei = lci + lni - 1$.  

138
Within this strip in rows, there is a scan for nonzero column blocks over the index $j_{col} = 1, n_{ktj} \times n_{krj}$. The following column parameters for each block are computed:

- the column region index $krj = (j_{col}-1)/n_{ktj} + 1$.
- the column datatype index $ktj = \text{mod}(j_{col}-1, n_{ktj}) + 1$.
- the starting index in columns for the block $lcj = krj \times n_{ktj} + ktj \times krj$.
- the extent in columns of the block $lnj = ktj \times lenj(ktj, krj)$.

A check is made to assure $\text{correlated}(irow, j_{col}) = .true.$; 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 $\text{gCprodx()}$:

```fortran
call 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, lnj,ktabj(lcj), qmj_x,qmj_y,qmj_z,qlj_x,qlj_y, nvecs, idxj, 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 $\text{list.err}(ir)$, 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:


Outside of the evaluation loop, there is a check for nonzero values of $\text{list.row}$. If any exist, a message is written to $\text{stderr}$.

After the calculation is complete, the temporary arrays $\text{list.err}$, $\text{list.row}$, and $Cxj$ are deallocated, and $\text{rec.Cxpy()}$ 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 $\text{gCprodx()}$:

```fortran
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,lnj,ktabj, & ier)
```

139
Table 34: Summary of arguments to `rec.Cxpy()`. The 'columns' arguments `n_krj`, ..., `qlj-y` are not shown for brevity.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind_mat</td>
<td>INTEGER</td>
<td>IN</td>
<td>Matrix Type</td>
</tr>
<tr>
<td>kind_cov</td>
<td>INTEGER</td>
<td>IN</td>
<td>Covariance Type</td>
</tr>
<tr>
<td>sparses</td>
<td>INTEGER</td>
<td>IN</td>
<td>Matrix Type</td>
</tr>
<tr>
<td>n_kri</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Regions (rows)</td>
</tr>
<tr>
<td>kr_loci</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Region Start (rows)</td>
</tr>
<tr>
<td>kr_leni</td>
<td>INTEGER(n_kr)</td>
<td>IN</td>
<td>Region Lengths (rows)</td>
</tr>
<tr>
<td>n_kti</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Data Types (rows)</td>
</tr>
<tr>
<td>kt_loci</td>
<td>INTEGER(n_kt,n_kr)</td>
<td>IN</td>
<td>Start of each kt block (rows)</td>
</tr>
<tr>
<td>kt_leni</td>
<td>INTEGER(n_kt,n_kr)</td>
<td>IN</td>
<td>Length of each kt block (rows)</td>
</tr>
<tr>
<td>n_i</td>
<td>INTEGER</td>
<td>IN</td>
<td>Attribute Array Dimension (rows)</td>
</tr>
<tr>
<td>ktabi</td>
<td>INTEGER(n_i)</td>
<td>IN</td>
<td>Level Index for Look-Up Tables (rows)</td>
</tr>
<tr>
<td>qri.x</td>
<td>REAL(n.i)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td>qri.y</td>
<td>REAL(n.i)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td>qri.z</td>
<td>REAL(n.i)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td>qmi.x</td>
<td>REAL(n.i)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td>qmi.y</td>
<td>REAL(n.i)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td>qmi.z</td>
<td>REAL(n.i)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td>qli.x</td>
<td>REAL(n.i)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_i$ (rows)</td>
</tr>
<tr>
<td>qli.y</td>
<td>REAL(n.i)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_i$ (rows)</td>
</tr>
<tr>
<td>nvecs</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of vectors</td>
</tr>
<tr>
<td>ldxj</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $x$</td>
</tr>
<tr>
<td>xj</td>
<td>REAL(ldxj,nvecs)</td>
<td>IN</td>
<td>Input vectors $x$</td>
</tr>
<tr>
<td>ldyi</td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $y$</td>
</tr>
<tr>
<td>yi</td>
<td>REAL(ldyi,nvecs)</td>
<td>OUT</td>
<td>Output vectors $y$</td>
</tr>
<tr>
<td>istorat</td>
<td>INTEGER</td>
<td>OUT</td>
<td>Return Status</td>
</tr>
</tbody>
</table>
The arguments of \texttt{gCprodx()} are summarized in Table 35.

The first step taken in \texttt{gCprodx()} is a set of sanity checks on its arguments:

- \( \text{ldXj} \geq \text{lenj} \) and \( \text{ldCxj} \geq \text{leni} \).
- \( \text{kind.cov} \in \{\text{kind.covF}, \text{kind.covS}, \text{kind.covV}\} \)

If any of these conditions are violated, \texttt{gCprodx()} returns with \( \text{ierr} = -1 \). If either the number of vectors \( \text{nvecs} \) or the block row dimension \( \text{leni} \) are nonpositive, \texttt{gCprodx()} returns. In the case of a block with \( \text{lenj} = 0 \), the resultant operator product \( \text{Cxj} \) for this block is zero:

\[
\text{Cxj}(1:1:\text{lenj},1:\text{nvecs}) = 0.
\]

The block of the error correlation operator is stored in dynamic memory in the array \( \text{Corr}(1:1:\text{lenj}) \). This block is calculated through a call to \texttt{offdcorF()}:

\[
\text{call offdcorF(kind.cov,} \quad \&
\begin{align*}
\text{kti,} & \quad \&
\text{leni,} & \quad \&
\text{qri.x,} & \quad \&
\text{qri.y,} & \quad \&
\text{qri.z,} & \quad \&
\text{qmi.x,} & \quad \&
\text{qmi.y,} & \quad \&
\text{qmi.z,} & \quad \&
\text{qli.x,} & \quad \&
\text{qli.y,} & \quad \&
\text{ktabi,} & \quad \&
\text{ktj,} & \quad \&
\text{lenj,} & \quad \&
\text{qrj.x,} & \quad \&
\text{qrj.y,} & \quad \&
\text{qrj.z,} & \quad \&
\text{qmj.x,} & \quad \&
\text{qmj.y,} & \quad \&
\text{qmj.z,} & \quad \&
\text{qlj.x,} & \quad \&
\text{qlj.y,} & \quad \&
\text{ktabj,} & \quad \&
\text{Mtyp}, & \quad \&
\text{Corr,} & \quad \&
\text{ierr} & \quad \).
\end{align*}
\]

The operator block is returned in the structure \( \text{Corr}(1:1:\text{lenj}) \). The type of block calculated is returned in the \texttt{CHARACTER} variable \( \text{Mtyp} \), which has value 'T' for a transposed block, 'N' for a non-transposed block, and 'E' in the event of an error.

If \( \text{Mtyp} \in \{\text{'T', 'r', 'N', 'n'}\} \), the application of the forecast error correlation operator block to each of the input vectors in \( \text{xj} \) is done via a set of calls to the BLAS routine \texttt{SGEMV}:

\[
\text{do ivec=1,nvecs}
\begin{align*}
\text{call SGEMV(Mtyp,} & \quad \&
\begin{align*}
\text{lenj,} & \quad \&
\text{leni,} & \quad \&
\text{Corr,} & \quad \&
\text{lenj,} & \quad \&
\text{xj(1,ivec),} & \quad \&
\text{1,} & \quad \&
\text{0.}, \quad \&
\text{Cxj(1,ivec),} & \quad \&
\text{1)
\end{align*}
\end{align*}
\text{end do.}
\]

The result of the product returned in \( \text{Cxj}(1:1:\text{ldCxj},1:\text{nvecs}) \).
Table 35: Summary of arguments to `gCprodx()`.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type/Dimensions</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>kind.cov</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Covariance Type</td>
</tr>
<tr>
<td><code>kri</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Region index (rows)</td>
</tr>
<tr>
<td><code>kti</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Datatype index (rows)</td>
</tr>
<tr>
<td><code>leni</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Row Length</td>
</tr>
<tr>
<td><code>ktabi</code></td>
<td>INTEGER(leni)</td>
<td>IN</td>
<td>Level Index (rows)</td>
</tr>
<tr>
<td><code>qri.x</code></td>
<td>REAL(leni)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td><code>qri.y</code></td>
<td>REAL(leni)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td><code>qri.z</code></td>
<td>REAL(leni)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td><code>qmi.x</code></td>
<td>REAL(leni)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td><code>qmi.y</code></td>
<td>REAL(leni)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td><code>qmi.z</code></td>
<td>REAL(leni)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td><code>qli.x</code></td>
<td>REAL(leni)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_l$ (rows)</td>
</tr>
<tr>
<td><code>qli.y</code></td>
<td>REAL(leni)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_l$ (rows)</td>
</tr>
<tr>
<td><code>krj</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Region index (rows)</td>
</tr>
<tr>
<td><code>ktj</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Datatype index (rows)</td>
</tr>
<tr>
<td><code>lenj</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Row Length</td>
</tr>
<tr>
<td><code>ktabj</code></td>
<td>INTEGER(lenj)</td>
<td>IN</td>
<td>Level Index (rows)</td>
</tr>
<tr>
<td><code>qr.j.x</code></td>
<td>REAL(lenj)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td><code>qr.j.y</code></td>
<td>REAL(lenj)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td><code>qr.j.z</code></td>
<td>REAL(lenj)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_r$ (rows)</td>
</tr>
<tr>
<td><code>qm.j.x</code></td>
<td>REAL(lenj)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td><code>qm.j.y</code></td>
<td>REAL(lenj)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td><code>qm.j.z</code></td>
<td>REAL(lenj)</td>
<td>IN</td>
<td>$z$-component of $\hat{e}_m$ (rows)</td>
</tr>
<tr>
<td><code>ql.j.x</code></td>
<td>REAL(lenj)</td>
<td>IN</td>
<td>$x$-component of $\hat{e}_l$ (rows)</td>
</tr>
<tr>
<td><code>ql.j.y</code></td>
<td>REAL(lenj)</td>
<td>IN</td>
<td>$y$-component of $\hat{e}_l$ (rows)</td>
</tr>
<tr>
<td><code>nvecs</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of vectors</td>
</tr>
<tr>
<td><code>ldxj</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $x$</td>
</tr>
<tr>
<td><code>xj</code></td>
<td>REAL(ldxj,nvecs)</td>
<td>IN</td>
<td>Input vectors $x$</td>
</tr>
<tr>
<td><code>ldCxj</code></td>
<td>INTEGER</td>
<td>IN</td>
<td>Leading dimension of $y$</td>
</tr>
<tr>
<td><code>Cj</code></td>
<td>REAL(ldyj,nvecs)</td>
<td>OUT</td>
<td>Output vectors $y$</td>
</tr>
<tr>
<td><code>ierr</code></td>
<td>INTEGER</td>
<td>OUT</td>
<td>Return Status</td>
</tr>
</tbody>
</table>
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.

```plaintext
subroutine getAIpuv( npieces, lat_list, lon_list, pres_list, &
  time_list, kx_list, kt_list, dials_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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>npieces</td>
<td>INTEGER</td>
<td>INOUT</td>
<td>Number of Innovations</td>
</tr>
<tr>
<td>lat_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Latitude</td>
</tr>
<tr>
<td>lon_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Longitude</td>
</tr>
<tr>
<td>pres_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Pressure (hPa)</td>
</tr>
<tr>
<td>time_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>$\Delta t$ (min) from Analysis Time</td>
</tr>
<tr>
<td>xx_list</td>
<td>INTEGER(npieces)</td>
<td>INOUT</td>
<td>Data Source Index $xx$</td>
</tr>
<tr>
<td>kt_list</td>
<td>INTEGER(npieces)</td>
<td>INOUT</td>
<td>Data Type Index $kt$</td>
</tr>
<tr>
<td>dels_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Innovation O-F</td>
</tr>
<tr>
<td>sigF_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Forecast Error Variances $\sigma_f$</td>
</tr>
<tr>
<td>sigO_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Observation Error Variances $\sigma_o$</td>
</tr>
<tr>
<td>im</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Longitudinal Grid Points</td>
</tr>
<tr>
<td>jnp</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Latitudinal Grid Points</td>
</tr>
<tr>
<td>psl_sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast Sea-Level Pressure Error Variances</td>
</tr>
<tr>
<td>usl_sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast $u_{sl}$ Error Variances</td>
</tr>
<tr>
<td>vsl_sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast $v_{sl}$ Error Variances</td>
</tr>
<tr>
<td>psl_inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Sea-Level Pressure Analysis Increments</td>
</tr>
<tr>
<td>usl_inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$u_{sl}$ Analysis Increments</td>
</tr>
<tr>
<td>vsl_inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$v_{sl}$ Analysis Increments</td>
</tr>
<tr>
<td>psl_sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Sea-Level Pressure Analysis Error Variances</td>
</tr>
<tr>
<td>usl_sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$u_{sl}$ Analysis Error Variances</td>
</tr>
<tr>
<td>vsl_sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$v_{sl}$ Analysis Error Variances</td>
</tr>
</tbody>
</table>
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 `getAlzuv()` via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>npieces</td>
<td>INTEGER</td>
<td>INOUT</td>
<td>Number of Innovations</td>
</tr>
<tr>
<td>lat.list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Latitude</td>
</tr>
<tr>
<td>lon.list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Longitude</td>
</tr>
<tr>
<td>pres.list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Pressure (hPa)</td>
</tr>
<tr>
<td>time_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>$\Delta t$ (min) from Analysis Time</td>
</tr>
<tr>
<td>kx.list</td>
<td>INTEGER(npieces)</td>
<td>INOUT</td>
<td>Data Source Index $k_x$</td>
</tr>
<tr>
<td>kt.list</td>
<td>INTEGER(npieces)</td>
<td>INOUT</td>
<td>Data Type Index $k_t$</td>
</tr>
<tr>
<td>dels_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Innovation O-F</td>
</tr>
<tr>
<td>sigF.list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Forecast Error Variances $\sigma_f$</td>
</tr>
<tr>
<td>sigO.list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Observation Error Variances $\sigma_o$</td>
</tr>
<tr>
<td>im</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Longitudinal Grid Points</td>
</tr>
<tr>
<td>np</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Latitudinal Grid Points</td>
</tr>
<tr>
<td>mlev</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Grid Vertical Levels</td>
</tr>
<tr>
<td>pres.lev</td>
<td>REAL(mlev)</td>
<td>INOUT</td>
<td>Pressure Levels</td>
</tr>
<tr>
<td>z.sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast Geopotential Height Error Variances</td>
</tr>
<tr>
<td>u.sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast $u$ Error Variances</td>
</tr>
<tr>
<td>v.sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast $v$ Error Variances</td>
</tr>
<tr>
<td>z.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Geopotential Height Analysis Increments</td>
</tr>
<tr>
<td>u.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$u$ Analysis Increments</td>
</tr>
<tr>
<td>v.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$v$ Analysis Increments</td>
</tr>
<tr>
<td>z.sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Geopotential Height Analysis Error Variances</td>
</tr>
<tr>
<td>u.sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$u$ Analysis Error Variances</td>
</tr>
<tr>
<td>v.sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>$v$ Analysis Error Variances</td>
</tr>
</tbody>
</table>
The Upper-air Water Vapor Mixing Ratio Analysis \texttt{getAImix}() The PSAS interface that computes only water vapor mixing ratio analysis increments is the routine \texttt{getAImix()}, whose interface is shown below.

\begin{verbatim}
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 & )
\end{verbatim}

The calling tree for \texttt{getAImix()} is shown in Figure 9, and its arguments are summarized in Table 38.

This routine is modified from \texttt{getAIall()} in the following way:

- In order to keep the same interface to some routines called in the analysis (e.g., \texttt{getAinc()}), some of the arrays are vestigial; that is, they are dimensioned as scalars. These arrays are: \texttt{psl.sigF(1,1,1)}, \texttt{usl.sigF(1,1,1)}, \texttt{vsl.sigF(1,1,1)}, \texttt{z.sigF(1,1,1)}, \texttt{u.sigF(1,1,1)}, \texttt{v.sigF(1,1,1)}, \texttt{psl.inc(1,1,1)}, \texttt{usl.inc(1,1,1)}, \texttt{vsl.inc(1,1,1)}, \texttt{z.inc(1,1,1)}, \texttt{uinc(1,1,1)}, \texttt{vin(1,1,1)}, \texttt{psl.sigA(1,1,1)}, \texttt{usl.sigA(1,1,1)}, \texttt{vsl.sigA(1,1,1)}, \texttt{z.sigA(1,1,1)}, \texttt{uslsigA(1,1,1)}, and \texttt{vslAgA(1,1,1)}.

- The LOGICAL parameters \texttt{want.psl}, \texttt{want.usl}, \texttt{want.vsl}, \texttt{want.z}, \texttt{want.u}, and \texttt{want.v} are set to .FALSE.

- Since no multivariate upper-air AIs are to be calculated, calls to the routines \texttt{dervsigF_slD()}, \texttt{dervsigF_slW()}, \texttt{dervsigF_upD()}, and \texttt{dervsigF_upW()} are not made.
Table 38: Data passed into `getAIMix()` via its interface.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>npieces</td>
<td>INTEGER</td>
<td>INOUT</td>
<td>Number of Innovations</td>
</tr>
<tr>
<td>lat_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Latitude</td>
</tr>
<tr>
<td>lon_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Longitude</td>
</tr>
<tr>
<td>pres_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Pressure (hPa)</td>
</tr>
<tr>
<td>time_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>$\Delta t$ (min) from Analysis Time</td>
</tr>
<tr>
<td>kx_list</td>
<td>INTEGER(npieces)</td>
<td>INOUT</td>
<td>Data Source Index $kx$</td>
</tr>
<tr>
<td>kt_list</td>
<td>INTEGER(npieces)</td>
<td>INOUT</td>
<td>Data Type Index $kt$</td>
</tr>
<tr>
<td>dels_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Innovation O-F</td>
</tr>
<tr>
<td>sigF_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Forecast Error Variances $\sigma_f$</td>
</tr>
<tr>
<td>sigO_list</td>
<td>REAL(npieces)</td>
<td>INOUT</td>
<td>Observation Error Variances $\sigma_o$</td>
</tr>
<tr>
<td>im</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Longitudinal Grid Points</td>
</tr>
<tr>
<td>jnp</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Latitudinal Grid Points</td>
</tr>
<tr>
<td>mlev</td>
<td>INTEGER</td>
<td>IN</td>
<td>Number of Grid Vertical Levels</td>
</tr>
<tr>
<td>pres_lev</td>
<td>REAL(mlev)</td>
<td>INOUT</td>
<td>Pressure Levels</td>
</tr>
<tr>
<td>mix.sigF</td>
<td>REAL(im,jnp)</td>
<td>IN</td>
<td>Forecast Mixing Ratio Error Variances</td>
</tr>
<tr>
<td>mix.inc</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Mixing Ratio Analysis Increments</td>
</tr>
<tr>
<td>mix.sigA</td>
<td>REAL(im,jnp)</td>
<td>OUT</td>
<td>Mixing Ratio Analysis Error Variances</td>
</tr>
</tbody>
</table>
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)
  |-lnblink(lnblink.f)
  |-pZETBEG(zeits.F)
    . |-syszeit(zeits.F)
    . . |-second(?)
    . . |-secondr(?)
    . . |-time(?)
    . . |-etime(?)
    . |-syszeit("")
    |-getenv(?)
    |-I90_LoadF(m_inpak90.f90)
      . |-opntext(opntext.F)
      . |-asunit(?)
      . |-i90_page(m_inpak90.f90)
      . |-i90_trim(m_inpak90.f90)
      . |-i90_pad(m_inpak90.f90)
    |-psasexit(psasexit.f)
      . |-pzeitpri(zeits.f)
      . . |-syszeit("")
      |-exit(?)
  |-pZETBEG("")
  |-GETDELE2(getdel2.f)
    . |-luavail("")
    . |-lnblink("")
    . |-LABLIN(m_inpak90.f90)
    . . |-I90_label(m_inpak90.f90)
    . . |-STRGET(m_inpak90.f90)
    . . |-I90_gstr(m_inpak90.f90)
    . . . |-I90_trim("")
    . . |-opniees(opniees.F)
    . . |-asunit("")
  |-pZEITEND(zeits.F)
    . |-syszeit("")
    |-psasexit("")
  |-iniainc(iniainc.f)
    . |-LABLIN("")
  |-iniaio(aio_grads.f)
    . |-LUAVAIL("")
    . |-amatch(amatch.f)
    . . |-lnblink("")
    . |-listvals(listvals.f)
    . |-psasexit("")
    . |-LABLIN("")
    . |-STRGET("")
    . |-psasexit("")
    . |-lnblink("")
    . |-lablin("")
```

149
DAO Ofice Note 1998-05

I -getwrd(m_inpak90.f90)
I -i90_gtoken(m_inpak90.f90)
I -i90_trim('')
I -getwrd('')
I -LABLIN('')
I -psasexit('')
I -LABLIN('')
I -psasexit('')
I -LABLIN('')
I -psasexit('')
I -opnieee('')
I -psasexit('')

I -psasexit('')
I -I90_Release(m_inpak90.f90)
I -zeitbeg('')
I -getAIall(getAIall.F)
I -snrm2('')
I -luavail('')
I -lnblnk('')
I -psasrcbd(psasrcbd.f)
I -zeitbeg('')
I -psasexit('')
I -getenv('')
I -zeitbeg('')
I -I90_LoadF('')
I -zeitend('')
I -psasexit('')
I -zeitbeg('')
I -initRSRC(initRSRC.F90)
I -qtrigO(qtrigO.f)
I -knameO(kname0.F)
I -rdkttbl(rdkttbl.f)
I -lnblnk('')
I -lablin('')
I -rdnext(m_inpak90.f90)
I -i90_gline(m_inpak90.f90)
I -getwrd('')
I -getstr(m_inpak90.f90)
I -i90_gstr('')
I -getwrd('')
I -rdnext('')
I -psasexit('')
I -kxnameO(kxname0.f)
I -rdkxtbl(rdkxtbl.f)
I -lnblnk('')
I -lablin('')
I -rdnext('')
I -getwrd('')
I -getstr('')
I -rdnext('')
I -psasexit('')
I -set_OEclas(set_OEclas.F90)
I -listvals('')
I -lstins(lstins.F)
I -lnblnk('')
I -tabSList(tabSList.f90)
I -psasexit('')
I -indexxs(indexxs.f90)
I -getTab_(tabllist.F90)
DAO Office Note 1998-05

I-nextTab_(tablist.F90)
I-getVal_(tablist.F90)
I-nextVal_(tablist.F90)
I-getTab_(^)
I-nextTab_(^)
I-nextVal_(^)
I-getTab_(^)
I-nextTab_(^)
I-nextVal_(^)
I-getVal_(^)
I-nextVal_(^)

I-inxSlist(inxSlist.F90)
I-rdlevels(^)
I-psasexit(^)
I-rdtabl(rdtabl.F90)
I-lnblank(^)
I-lablin(^)
I-rdnext(^)
I-getwrd(^)
I-getstr(^)
I-rdnext(^)

I-set_OEhCor(set_OEhCor.F90)
I-tabSlist(^)
I-inxSlist(^)
I-rdpar(rdpar.F90)
I-lablin(^)
I-rdnext(^)
I-getwrd(^)
I-getstr(^)
I-rdnext(^)

I-set_OEvCor(set_OEvCor.F90)
I-tabSlist(^)
I-inxSlist(^)
I-rdvctbl(rdvctbl.F90)
I-lnblank(^)
I-lablin(^)
I-rdnext(^)
I-getwrd(^)
I-getstr(^)
I-rdnext(^)

I-set_FEhCor(set_FEhCor.F90)
I-rdpar(^)
I-psasexit(^)
I-rdpar(^)
I-psasexit(^)
I-rdpar(^)
I-psasexit(^)
I-rdpar(^)
I-psasexit(^)
I-rdpar(^)
I-psasexit(^)
I-rdpar(^)
I-psasexit(^)
I-rdpar(^)
I-psasexit(^)
I-rdpar(^)
I-psasexit(^)
I-rdPars(^)

I-tabl_FEsigW(tabl_FEsigW.F90)
I-rdpar(^)
I-psasexit(^)
I-rdPars(^)
I-tabl_FEalpha(tabl_FEalpha.F90)
I-rdPars(^)
DAO Office Note 1998-05

I-ipick(ipick.f)
I-INDEXXI(indexxi.f)
I-PERMUTI(^)
I-PERMUTR(^)
I-psasexit(^)
I-typsort(typsort.F)
I-psasexit(^)
I-INDEXXI(^)
I-PERMUTI(^)
I-PERMUTR(^)
I-indexxi(^)
I-PERMUTI(^)
I-psasexit(^)
I-INDEXSR(indexsr.f)
I-PERMUTI(^)
I-psasexit(^)
I-PERMUTR(^)
I-INDEXSR(indexsr.f)
I-psasexit(^)
I-PERMUTI(^)
I-PERMUTR(^)
I-pzeitend(^)
I-pzeitbeg(^)
I-dupelim(dupelim.f)
I-TOFRONT(^)
I-pzeitend(^)
I-pzeitbeg(^)
I-proxel(proxel.f)
I-sepang(sepang.f)
I-psasexit(^)
I-LL2XYZ(^)
I-psasexit(^)
I-PRXSOB(proxel.f)
I-psasexit(^)
I-TOFRONT(^)
I-pzeitend(^)
I-gridxx0(gridxx.f)
I-LABLIN(^)
I-OBSSMRY(^)
I-obstat(^)
I-pZEITBEG(^)
I-merg_plevs(merg_plevs.F90)
I-psasexit(^)
I-indexxi(^)
I-permuti(^)
I-merg_plevs(^)
I-merg_lats(merg_lats.F90)
I-psasexit(^)
I-tabRlist(tabRlist.F90)
I-psasexit(^)
I-tabRlist(tabRlist.F90)
I-psasexit(^)
I-merg_plevs(^)
I-indexxi(^)
I-getTab(_)
I-nextTab(_)
I-getVal(_)
I-nextVal(_)
I-getTab(_)
I-nextTab(_)
I-getVal(_)
I-nextVal(_)
I-getTab(_)
I-nextTab(_)
I-getVal(_)
I-nextVal(_)
I-tabRlist(^)
I-set_fechh(set_fechh.F90)
I-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(')
  . . . |PSASi11_sigW_a(imat_sigW.F90)
  . . . |psasexit(')
  . . . |slogtab(')
  . . . |PSASi11_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() |
| -GRADSC(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() |
| -GRADS(m_grads.f90) |
| -pzeitend() |
| -pzeitbeg() |
| -dervsigF_s1D(dervsigF_s1D.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() |
| -112qvec(112qvec.f90) |
| -setpix(setpix.f90) |
| -slogtab() |
| -slintab(slintab.f90) |
| -vindex() |
| -op_Mx(op_Mx.F90) |
| -sparse(sparse.f90) |
| -sepang() |
DAO Office Note 1998-05

\[\begin{align*}
&\text{snrm2}(\cdot) \\
&\text{psasexit}(\cdot) \\
&\text{covUxpy}(...) \\
&\text{mv\_diag}(\text{mv\_diag.f90}) \\
&\text{sym\_Cpxy}(\text{sym\_Cpxy.F90}) \\
&\text{sparse}(\cdot) \\
&\text{snrm2}(\cdot) \\
&\text{pzeitbeg}(\cdot) \\
&\text{pzeitend}(\cdot) \\
&\text{Cprod}(\text{Cprod.f90}) \\
&\text{diagcorF}(\text{cordriv.f90}) \\
&\text{SSPMV}(\cdot) \\
&\text{pzeitend}(\cdot) \\
&\text{Cprodx}(\text{Cprod.f90}) \\
&\text{psasexit}(\cdot) \\
&\text{mv\_diag}(\cdot) \\
&\text{sym\_Cpxy}(\cdot) \\
&\text{psasexit}(\cdot) \\
&\text{mv\_diag}(\cdot) \\
&\text{Cprod}(\text{Cprod.f90}) \\
&\text{diagcorF}(\text{cordriv.f90}) \\
&\text{diagonal}(\cdot) \\
&\text{Cprodx}(\text{Cprodx.f90}) \\
&\text{psasexit}(\cdot) \\
&\text{mv\_diag}(\cdot) \\
&\text{sym\_Cpxy}(\cdot) \\
&\text{psasexit}(\cdot) \\
&\text{mv\_diag}(\cdot) \\
&\text{Cprod}(\text{Cprod.f90}) \\
&\text{diagcorF}(\text{cordriv.f90}) \\
&\text{SSPMV}(\cdot) \\
&\text{pzeitend}(\cdot) \\
&\text{Cprod}(\text{Cprod.f90}) \\
&\text{diagonal}(\cdot) \\
&\text{SSPMV}(\cdot) \\
&\text{pzeitend}(\cdot) \\
&\text{getivec}(\cdot) \\
&\text{aj\_Gam}(\text{aj\_Gam.f90}) \\
&\text{mv\_Gam}(\text{mv\_Gam.f90}) \\
&\text{pzeitend}(\cdot) \\
&\text{solve4x}(\text{solve4x.F})
\end{align*}\]
-snrm2(*)
-psasexit(*)
-ll2qvec(*)
-setpix(*)
-logtab(*)
-slintab(*)
-conjgr(conjgr.F)
-sdot(*)
-snrm2(*)
-pzeitbeg(*)
-SCOPY(*)
-SCOPY(*)
-conjgr2(conjgr.F)
-sparse(*)
-sdot(*)
-snrm2(*)
-pzeitbeg(*)
-SCOPY(*)
-pzeitbeg(*)
-conjgr1(conjgr.F)
-sdot(*)
-snrm2(*)
-linblnk(*)
-luavail(*)
-SCOPY(*)
-diagcorM(diagcorM.F90)
-psasexit(*)
-diagcorD(*)
-check_istat_(diagcorM.F90)
-add_corM_(diagcorM.F90)
-add_smatt(diagcorM.F90)
-add_iden_(diagcorM.F90)
-add_diag_(diagcorM.F90)
-diagcorU(*)
-check_istat_(*)
-add_corM_(*)
-diagcorD(diagcorD.f90)
-fDDcori(cormats.f90)
-fDDcorx(cormats.f90)
-fDDcori(*)
-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(*)
-SCOPY(*)
-cgnorm(cgnorm.f)
DAO Office Note 1998-05

I-pzeitend(-)
-I-pzeitbeg(-)
| wrtaio(aio_grds.f)
. |-psasexit(-)
. |-WTBUF(aio_grds.f)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
. |-WTBUF(-)
. |-psasexit(-)
Appendix C: A Sample Resource File psas.rc

# PSAS.header.rc
#
# //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# //
# # A little bit of the history:
# # 01Oct96 - J. Gun - committed version 103 with G02.5 alike data
# # 13Aug96 - J. Gun - committed version 102 with modified correlation
# # function for FcstErr and ObsErr for tevA in
# # v100.
# # 04Aug96 - J. Gun - committed version 101
# # 04Aug96a - J. Gun - committed version 100
# # 04Aug96 - J. Gun - Put under CVS control
# # 21May96 - J. Gun - Included tuned statistics of v100.
# #
# # 05Oct95 - Jing G. - New vertical correlation functions. Earlier
# # tables are not positive definite.
# #
# # + FcstErrCor11.mk: the 14 level table was PD,
# # but not the 18 level table;
# #
# # + ObsErrmCor11.mk: tables in for both 14
# # and 18 levels are either not PD or poorly
# # conditioned.
# #
# # - Added comments for some earlier changes:
# # + Precondition conjgrit() convergence rate
# # + New entries
# #
# 13Jun96 - Jing G. - Fixed an error in the earlier version with
# all data above 30 mb excluded.
# 31Mar96 - Jing G. - Created a version with G025.1.0 and original
# PSAS, with 18 levels error statistics.
# 04Feb95 - Jing G. - Added time box for setup()
# - Edited for generic libpsas.a calls.
# - Removed some blank in one of entry, 'del,leg...
# for...'; both in this file and in prxes1()
# - It may cause some backward compatibility prog-
# ram. Since this is the first version to be
# complied for libpsas.a, it is better to make
# it right.
# 06Jan95 - Jing G. - Added ks-kt boxes
# 30Dec94 - Jing G. - Removed Analysis-Increment-Computation-
# Restrictions
# - Baking a test for used level wind analysis
# file: psas.rc - last change: 04Dec94 jing g. - added option to
# added option to restrict data included in eis
# computations.
# file: psas.rc - last change: 04Dec94 (A. da S.)
#
# This is an IMPAR77 style resource file for input of PSAS parameters.
#
#
# //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PSAS_test.rsrc
#
# INPUT/OUTPUT FILE NAMES
#
# innovation_file_name: /data/se.dat1.D00.r00.xu
# analysis_increment_output_file_name: EXP1D04_DATE_DATE_TIME_grid
# feat.err_grads_desc_file: <>
#
# Only inquire at 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 925 850 700 500 400 300 200 150 100 70 50 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

# 0 levels
# Analysis_incs: Levels: Pressure .01 10 200 500
# Analysis_incs: Levels: Pressure 500
#
# ANALYSIS INCREMENT OUTPUT
#
# want_analysis_incs_of_sea_level_u: yes # yes or no
# want_analysis_incs_of_sea_level_v: yes # yes or no
# want_analysis_incs_of_sea_level_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_mix: yes # yes or no
#
# PSAS.rum.rsrc

# Equal-area intermediate grid settings
latitude_threshold_for_equal_area_grid: 45

# PROXIMITY CHECK PARAMETERS

radius_in_km_for_proximity_check: 300. # km
163

```
del_inp for vertical proximity check: O.1 # del inp
do you want detailed superb list: n # y or n

# CONVERGENT GRADIENT PARAMETERS

# The next entry should be removed.
method_ef_solution: 1 # 0 direct solver
                   # 1 conjugate gradient

level_for_banded_approximation: 5 # 4 neglect crr x > 3000 km
                                 # 5 neglect crr x > 6000 km
                                 # (actual distances depend
                                 # on seplm below).

# The next entry should be removed.
frequency_to_output_analysis_inc_during_cg: 999 # 1 every iteration,
                                          # 2 every other
                                          # 9999 never

# Levels (*) --> 1 2 3 4 5
#   ---------- ---------- ---------- ----------
conjrseparation_limits: 0 0 0 26.5 59.26 # seplm
conjrmaximal_ne_iterations: 32 16 1 8 8 # maxpass
conjrminimum_ne_iterations: 16 8 1 4 4 # minpass
conjr_max_tolerances: 0.1 0.1 0.1 0.1 0.1 # criter
conjr_min_tolerances: 0.3 0.3 0.3 0.3 0.3 # criter
conjr_tolerances: 0.1 0.1 0.1 0.1 0.1 # criter
conjr_Rsmall: 16 # nsml
conjr_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.

# DataBases:
#
# NASA/GSFC, Data Assimilation Office, Code 910.3, GHGS/ODS
# #
# # NAME: DataBases: - Data selection tables in 6-d boxes
# # REVISED HISTORY:
# # 09/88 - J. Gu - implemented to replace the original
# # 09/89 - J. Gu - replaced 'regional boxes' resources.
# #
# # KX KX LAT LEN PRES TIME
# #```
DAO Office Note 1998-05

Version 1 Dated 12/06/1999

1 32 1 7 -90 +90 -180 +180 .009 1000.1 -180 +180

---

# Levels interpolated from NRC/NESSIS-TUVS 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

# net 3
# 5
33 56 1 7 -90 +90 -180 +180 4.9 5.1 -180 +180

# net 7
# 10
33 56 1 7 -90 +90 -180 +180 9.9 10.1 -180 +180

# net 20
# 30
# 50
# 70
# 100
33 56 1 7 -90 +90 -180 +180 29.9 100.1 -180 +180

# net 150
# 200
33 56 1 7 -90 +90 -180 +180 199.9 200.1 -180 +180

# net 250
# 300
# 400
# 500
# 700
# 850
33 56 1 7 -90 +90 -180 +180 850.1 -180 +180

---

# Table 87 1 7 -90 +90 -180 +180 .009 1000.1 -180 +180

---

# Data Types Table.rsrc

#---------------------------------------------

# 09Jan96 - Jing C. - Tabulated version of "kt"
# xk : Original assigned sequence numbers, integers.
# name : Assigned variable names (used in ObsErr tables).
# unit : Units of the variables expected in the program.
# descr : Descriptions of the variables
# mvar : Multi-variate flags
#
# DatTypeTable:
#
# mvars unit descr
# xk name
# 1 u,SeaLEV m/sec Sea Level East-West Wind
# 2 v,SeaLEV m/sec Sea Level South-North Wind
# 3 p,SeaLEV hPa Sea Level Pressure
# 4 u,UprAir m/sec Upper Air East-West Wind
# 5 v,UprAir m/sec Upper Air South-North Wind
# 6 h,UprAir m Upper Air Geopotential Height
# 7 q,UprAir g/kg Upper Air Water Vapour Mixing Ratio

# This file specifies forecast error covariance parameters in the same
# way as the G1.5.0 run of September '96.
# Notes:
#
# moisture forecast errors:
# G1 analyzes relative humidity while FGGE 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 G1 run, since our tuning results are not very
# different.
#
# vertical correlations:
# vertical correlations have been slightly adjusted to ensure
# positive definiteness.
# Dick Dee 27Sep96
#

FestErr*9Cer,HE:

DAOH-COSINE GEO5-DA5/G1 pseudo multi-level horizontal correlations

# pres d1 c 1 c2 c3 c4 c5
1000 0.0340 0.0339 0.0316 0.0210 0.00105 1.20815
8 0.0340 0.0339 0.0316 0.0210 0.00105 1.20815

FestErr*9Cer,QC:

DAOH-COSINE GEO5-DA5/G1 pseudo multi-level horizontal correlations
```plaintext
# press d_1 c1 d2 d3 c4 c5
1000 0.030 .0 .0 1.0 .00216 1.9558
.4 0.030 .0 .0 1.0 .00216 1.9558

::

FestErrvEro_Crربية

# press 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 .49 .59 .80 .85 1.
250 .25 .40 .49 .72 .77 .86 1.
200 .18 .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 .05 .19 .24 .73 1.
60 .09 .13 .10 .08 .06 .01 .16 .17 .30 .67 .84 1.
50 .00 .00 .00 .00 .00 .00 .00 .06 .17 .32 .66 .80 1.
40 .00 .00 .00 .00 .00 .00 .00 .00 .01 .06 .18 .34 .63 .81 1.
30 .00 .00 .00 .00 .00 .00 .00 .00 .00 .01 .04 .19 .35 .63 .81 1.
20 .00 .00 .00 .00 .00 .00 .00 .00 .00 .00 .01 .02 .02 .18 .36 .84 .81 1.
10 .00 .00 .00 .00 .00 .00 .00 .00 .00 .00 .00 .01 .02 .02 .18 .36 .84 .81 1.
.4 .00 .00 .00 .00 .00 .00 .00 .00 .00 .00 .02 .04 .18 .33 .68 .82 1.

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

FestErrvEro_0.01

# press 1000 850 700 500 400 300 250 200 150 100 70 50 30 10 5 2 1 .4
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 0.04 0.09 0.18 0.61 1.00
300 0.04 0.03 0.22 0.50 1.00
250 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 1.00
150 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 1.00
50 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 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 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 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 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 1.00
.4 0.00 0.00 0.00 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

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

FestErrvSigma_Hind:
```
# FcstErrKer_SigWind: Fall '94 NA raob multivariate O-F's (omega0)
# DESCRIPTION: Forecast wind error standard deviations: stream function and
# velocity potential components.
# Tuned to fall 1994 North-American raob code height-wind
# O-F's obtained from the omega0 run.
# Horizontal correlations modeled by the compactly
# supported spline function.
# REVISED HISTORY:
# May 13, 1996 - C. Redder and D. Dee - (original table)

NULL

# model* desc

<table>
<thead>
<tr>
<th>press</th>
<th>sigx</th>
<th>sigy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>5.00</td>
<td>12.45</td>
</tr>
<tr>
<td>925</td>
<td>5.71</td>
<td>11.52</td>
</tr>
<tr>
<td>850</td>
<td>5.62</td>
<td>9.59</td>
</tr>
<tr>
<td>700</td>
<td>6.14</td>
<td>7.30</td>
</tr>
<tr>
<td>500</td>
<td>7.61</td>
<td>6.60</td>
</tr>
<tr>
<td>400</td>
<td>8.34</td>
<td>10.92</td>
</tr>
<tr>
<td>300</td>
<td>10.13</td>
<td>12.19</td>
</tr>
<tr>
<td>250</td>
<td>12.69</td>
<td>12.82</td>
</tr>
<tr>
<td>200</td>
<td>15.88</td>
<td>11.67</td>
</tr>
<tr>
<td>150</td>
<td>19.06</td>
<td>9.91</td>
</tr>
<tr>
<td>100</td>
<td>21.11</td>
<td>8.14</td>
</tr>
<tr>
<td>70</td>
<td>22.35</td>
<td>7.10</td>
</tr>
<tr>
<td>60</td>
<td>24.00</td>
<td>6.74</td>
</tr>
<tr>
<td>40</td>
<td>24.98</td>
<td>7.60</td>
</tr>
<tr>
<td>30</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>20</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>15</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>10</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>7</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>5</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>3</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>2</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>1</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>0.7</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>0.5</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>0.4</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>0.2</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>0.05</td>
<td>25.95</td>
<td>8.46</td>
</tr>
<tr>
<td>0.01</td>
<td>25.95</td>
<td>8.46</td>
</tr>
</tbody>
</table>
# NAME: FcstErr+Corr_VV:: Fall '94 Bi multivariate O-P's (emessaF)
# DESCRIPTION: Forecast wind error horizontal correlations: velocity potential
# component.
# Tuned to fall 1994 North-American rawinsonde height-wind
# O-P's obtained from the emessaF run.
# # REVISION HISTORY:
# May 13, 1996 - C. Redder and D. Dee - (original table)
#-----------------------------------------------

#press 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 0.30 0.20 0.10 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 0.00 1.00
300 0.00 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 0.00 1.00
200 0.00 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 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 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 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 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 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 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 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 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 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 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 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 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 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 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

# press 1000 925 850 700 500 400 300 250 200 150 100 70 50 40 30 20 15 10 7 5 3 2
# 1.00 0.70 0.50 0.40 0.30 0.20 0.10 0.05 0.01
::

FcstErr+hCor_VV::
# NAME: FcstErr+hCor_VV:: Fall '94 Bi multivariate O-P's (emessaF)
# DESCRIPTION: Forecast wind error horizontal correlations: velocity potential
# component.
# Tuned to fall 1994 North-American rawinsonde height-wind
# O-P's obtained from the emessaF run.
# Horizontal correlations modeled by the compactly
# supported spline function.
# Constant length scale --> separable correlation model.
#
# !REVISED HISTORY!
# Rev 13. 1996 - C. Redeker and P. Dee - (original table)
# Aug 14, 1996 - D. Dee - replace tuned length scales by a constant
# (tuned 500mb value)
#
# model: desc

GFS/VAM-CRES multi-level spline horizontal correlations
#
# press  d_m  L (km)
#
1000  3000  570.0
925   3000  570.0
850   3000  570.0
700   3000  570.0
600   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
60    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
0.1   3000  570.0
0.5   3000  570.0
0.2   3000  570.0
0.05  3000  570.0
0.01  3000  570.0

FcsvErrvCor_VF:
#
# NASA/GSFC. Data Assimilation Office. Code 910.3. CDES/DAS
#
# !NAME: FcsvErrvCor_VF: Fall '94 WI multivariate 0-7's (emaseAF)
# !DESCRIPTION: Forecast wind error vertical correlations: velocity potential
# component.
# Tuned to fall 1994 North-American reanalysis height-wind
# 0-7's obtained from the emaseAF run.
# Revision History:
# May 13, 1996 - C. Redder and S. 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
1.00 0.70 0.50 0.40 0.30 0.20 0.10

<table>
<thead>
<tr>
<th>Pres</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>925</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>850</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>700</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>500</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>400</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>300</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>250</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>200</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>150</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>100</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>50</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>40</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>30</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.70</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.50</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Note: This is a parameter table of mass-wind balance. It describes the pattern of the mass-wind balanced scheme. Version 1.00 - J. Sue - 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.
# This file specifies observation error covariances in the same way as the
# G1-2.5 run of September '96, with the following exceptions:
# moisture observation errors Ox = 7, 9, 10:
# G1 analyses relative humidity while PSAS uses mixing
# ratio. The statistical parameters in this file are
# for mixing ratio errors, obtained by tuning to rawinsonde
# O-P's over Europe (without removing the time mean).
# WRRHE height retrievals over ocean Ox = 39-44).
# in the G1 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
#
# **REVISEHISTORY:**
# 04Jan95 - Jing G. - Tabulated version of "km"
#
# **km** : Original assigned numbers of data sources. Int.
# **class** : Original char*8 names for compatibility and
# observation error class.
# **rank** : Assigned ranks, integer.
# **desc** : A long name or description, char*25.
#
<table>
<thead>
<tr>
<th>km</th>
<th>class</th>
<th>rank</th>
<th>desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SPC_LND</td>
<td>1000</td>
<td>Surface Land Obs - 1</td>
</tr>
<tr>
<td>2</td>
<td>SPC_LND</td>
<td>10100</td>
<td>Surface Land Obs - 2</td>
</tr>
<tr>
<td>3</td>
<td>SPC_SHF</td>
<td>10400</td>
<td>Surface Ship Obs - 1</td>
</tr>
<tr>
<td>4</td>
<td>SPC_SHF</td>
<td>10500</td>
<td>Surface Ship Obs - 2</td>
</tr>
<tr>
<td>5</td>
<td>MKY_BBF</td>
<td>10200</td>
<td>Environment Buoy</td>
</tr>
<tr>
<td>6</td>
<td>MKY_BBF</td>
<td>10300</td>
<td>Drifting Buoy</td>
</tr>
<tr>
<td>7</td>
<td>RAWINSE</td>
<td>100</td>
<td>Rawinsonde</td>
</tr>
<tr>
<td>8</td>
<td>PILOT_WD</td>
<td>-700</td>
<td>Pilot Balloon Wind</td>
</tr>
<tr>
<td>9</td>
<td>NAVRODS</td>
<td>-200</td>
<td>Ship Released Rawinsonde</td>
</tr>
<tr>
<td>10</td>
<td>DROPNDE</td>
<td>-500</td>
<td>Drapersonde</td>
</tr>
<tr>
<td>11</td>
<td>RSNDRNDE</td>
<td>-200</td>
<td>Radar - tracked Rawinsonde</td>
</tr>
<tr>
<td>12</td>
<td>ROCKETNDE</td>
<td>-400</td>
<td>Rocketonde</td>
</tr>
<tr>
<td>13</td>
<td>BALLOON</td>
<td>-900</td>
<td>Balloon</td>
</tr>
<tr>
<td>14</td>
<td>AER_LAND</td>
<td>-3000</td>
<td>Aircraft - Air/Sat Relay</td>
</tr>
<tr>
<td>15</td>
<td>AER_LAND</td>
<td>-3100</td>
<td>Aircraft - Int. Data Sys</td>
</tr>
<tr>
<td>16</td>
<td>AER_LAND</td>
<td>-5000</td>
<td>Aircraft Report</td>
</tr>
<tr>
<td>17</td>
<td>AER_LAND</td>
<td>-3300</td>
<td>Aircraft Coded Report</td>
</tr>
<tr>
<td>18</td>
<td>AER_LAND</td>
<td>-3400</td>
<td>Aircraft - ALP</td>
</tr>
<tr>
<td>#</td>
<td>Code</td>
<td>ID</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>--------</td>
<td>-------</td>
<td>--------------------</td>
</tr>
<tr>
<td>19</td>
<td>CLOTRVW</td>
<td>-3500</td>
<td>Cld Trk Wind - Wisc El</td>
</tr>
<tr>
<td>20</td>
<td>CLOTRVW</td>
<td>-3600</td>
<td>Cld Trk Wind - Wisc El</td>
</tr>
<tr>
<td>21</td>
<td>CLOTRVW</td>
<td>-3700</td>
<td>Cld Trk Wind - Wisc W</td>
</tr>
<tr>
<td>22</td>
<td>CLOTRVW</td>
<td>-3900</td>
<td>Cld Trk Wind - Wisc Ocean</td>
</tr>
<tr>
<td>23</td>
<td>CLOTRVW</td>
<td>-3900</td>
<td>Cld Trk Wind - Wisc Ocean</td>
</tr>
<tr>
<td>24</td>
<td>CLOTRVW</td>
<td>-4000</td>
<td>Cld Trk Wind - Wisc Ocean</td>
</tr>
<tr>
<td>25</td>
<td>CLOTRVW</td>
<td>-4100</td>
<td>Cld Trk Wind - Wisc Ocean</td>
</tr>
<tr>
<td>26</td>
<td>CLOTRVW</td>
<td>-4200</td>
<td>Cld Trk Wind - Wisc Ocean</td>
</tr>
<tr>
<td>27</td>
<td>CLOTRVW</td>
<td>-4300</td>
<td>Cld Trk Wind - Wisc Ocean</td>
</tr>
</tbody>
</table>

**# 28 SEASAT - 10000 SEASAT - Scatterometer**

**# 29 WSAT_55 - 660 WSAT 55**

**# 30 WSAT_57 - 610 WSAT 57**

**# 31 LIDAR - 1100 LIDAR Infrared Rad. - Strat**

**# 32 YPPA_55 - 10000 User-defined instrument 1**

**# 33 TRS4_NHL | 900 RESDIS NH Land AH type A**

**# 34 TRS5_NHL | 901 RESDIS SN Land AH type A**

**# 35 TRS6_NHL | 902 RESDIS NH Land AH type B**

**# 36 TRS7_NHL | 903 RESDIS SH Land AH type B**

**# 37 TRS8_NHL | 904 RESDIS NH Land AH type C**

**# 38 TRS9_NHL | 905 RESDIS SH Land AH type C**

**# 39 TRS10_NHL | 906 RESDIS NH Ocn AH type A**

**# 40 TRS11_NHL | 907 RESDIS SH Ocn AH type A**

**# 41 TRS12_NHL | 908 RESDIS NH Ocn AH type B**

**# 42 TRS13_NHL | 909 RESDIS SH Ocn AH type B**

**# 43 TRS14_NHL | 910 RESDIS NH Ocn AH type C**

**# 44 TRS15_NHL | 911 RESDIS SH Ocn AH type C**

**# 45 NOA4_NHL | 912 RESDIS NH Land PH type A**

**# 46 NOA5_NHL | 913 RESDIS NH Land PH type A**

**# 47 NOA6_NHL | 914 RESDIS NH Land PH type B**

**# 48 NOA7_NHL | 915 RESDIS NH Land PH type B**

**# 49 NOA8_NHL | 916 RESDIS NH Land PH type C**

**# 50 NOA9_NHL | 917 RESDIS SN Land PH type C**

**# 51 NOA10_NHL | 918 RESDIS NH Ocn PH type A**

**# 52 NOA11_NHL | 919 RESDIS SH Ocn PH type A**

**# 53 NOA12_NHL | 920 RESDIS NH Ocn PH type B**

**# 54 NOA13_NHL | 921 RESDIS SH Ocn PH type B**

**# 55 NOA14_NHL | 922 RESDIS NH Ocn PH type C**

**# 56 NOA15_NHL | 923 RESDIS SH Ocn PH type C**

**# 57 SPEA_NHL | 924 Special Sat NH - Ocn A**

**# 58 SPEB_NHL | 925 Special Sat SH - Ocn A**

**# 59 SPEC_NHL | 926 Special Sat NH - Ocn B**

**# 60 SPEE_NHL | 927 Special Sat SH - Ocn B**

**# 61 SPEF_NHL | 928 Special Sat NH - Ocn C**

**# 62 SPEG_NHL | 929 Special Sat SH - Ocn C**

**# 63 VASA_NHL | 1200 VAS NH Land - type A**

**# 64 VASA_SHL | 1201 VAS SN Land - type A**

**# 65 VASB_NHL | 1202 VAS NH Land - type B**

**# 66 VASB_SHL | 1203 VAS SN Land - type B**

**# 67 VASA_NHL | 1204 VAS NH Ocean- type A**

**# 68 VASA_SHL | 1205 VAS SN Ocean- type A**

**# 69 VASB_NHL | 1206 VAS NH Ocean- type B**

**# 70 VASB_SHL | 1207 VAS SN Ocean- type B**

**# 71 GLAA_NHL | 1000 NASA-5500 NH Land type A**

**# 72 GLAA_SHL | 1001 NASA-5500 SN Land type A**

**# 73 GLAB_NHL | 1002 NASA-5500 NH Land type B**

**# 74 GLAB_SHL | 1003 NASA-5500 SN Land type B**

**# 75 GLAC_NHL | 1004 NASA-5500 NH Land type C**

**# 76 GLAC_SHL | 1005 NASA-5500 SN Land type C**

**# 77 GLAD_NHL | 1006 NASA-5500 NH Land type D**

**# 78 GLAD_SHL | 1007 NASA-5500 SN Land type D**

**# 79 GLAE_NHL | 1008 NASA-5500 NH Ocean type A**
<table>
<thead>
<tr>
<th>Variable</th>
<th>u Sealev</th>
<th>v Sealev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.8</td>
<td>2.8</td>
</tr>
<tr>
<td>OBSEX=BUOY_REV:</td>
<td># lx = 5</td>
<td></td>
</tr>
<tr>
<td>p Sealev</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>u Sealev</td>
<td>2.8</td>
<td>2.8</td>
</tr>
<tr>
<td>OBSEX=BUOY_DEF:</td>
<td># lx = 6</td>
<td></td>
</tr>
<tr>
<td>p Sealev</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>u Sealev</td>
<td>2.8</td>
<td>2.8</td>
</tr>
<tr>
<td>OBSEX=RKNINSEDC:</td>
<td># lx = 7</td>
<td></td>
</tr>
<tr>
<td>q UpAir</td>
<td>0.76</td>
<td>0.71</td>
</tr>
<tr>
<td>w UpAir</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td>v UpAir</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td>H UpAir</td>
<td>4.01</td>
<td>5.10</td>
</tr>
<tr>
<td># vcor Q0:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>vcor RH:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OBSEX=DRUVMD:</td>
<td># lx = 8</td>
<td></td>
</tr>
<tr>
<td>u UpAir</td>
<td>4.4</td>
<td>5.2</td>
</tr>
<tr>
<td>v UpAir</td>
<td>4.4</td>
<td>5.2</td>
</tr>
<tr>
<td># vcor Q0:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>vcor RH:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OBSEX=RAVAPPS:</td>
<td># lx = 9</td>
<td></td>
</tr>
<tr>
<td>q UpAir</td>
<td>0.76</td>
<td>0.71</td>
</tr>
<tr>
<td>w UpAir</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td>v UpAir</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td>H UpAir</td>
<td>4.01</td>
<td>5.10</td>
</tr>
<tr>
<td># vcor Q0:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>vcor RH:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OBSEX=RKMPS:</td>
<td># lx = 10</td>
<td></td>
</tr>
<tr>
<td>u UpAir</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td>v UpAir</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td>H UpAir</td>
<td>4.01</td>
<td>5.10</td>
</tr>
<tr>
<td># vcor Q0:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>vcor RH:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OBSEX=RADVMD:</td>
<td># lx = 11</td>
<td></td>
</tr>
<tr>
<td>q UpAir</td>
<td>0.76</td>
<td>0.71</td>
</tr>
<tr>
<td># vcor Q0:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>vcor RH:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OBSEX=RGCMDSE:</td>
<td># lx = 12</td>
<td></td>
</tr>
<tr>
<td>H UpAir</td>
<td>3.75</td>
<td>5.2</td>
</tr>
<tr>
<td>u UpAir</td>
<td>5.2</td>
<td>6.3</td>
</tr>
<tr>
<td>v UpAir</td>
<td>5.2</td>
<td>6.3</td>
</tr>
<tr>
<td># vcor Q0:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>vcor RH:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OBSEX=RAVDMD:</td>
<td># lx = 13</td>
<td></td>
</tr>
<tr>
<td>u UpAir</td>
<td>2.2</td>
<td>2.2</td>
</tr>
<tr>
<td>v UpAir</td>
<td>2.2</td>
<td>2.2</td>
</tr>
<tr>
<td># vcor Q0:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>vcor RH:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OBSEX=RAVMDMD:</td>
<td># lx = 14</td>
<td></td>
</tr>
<tr>
<td>u UpAir</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td>v UpAir</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td># vcor Q0:</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>vcor RH:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>u_UprAir</td>
<td>u.u</td>
<td>v_UprAir</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
<td>----------</td>
</tr>
<tr>
<td>1.7</td>
<td>2.0</td>
<td>2.2</td>
</tr>
<tr>
<td>2.9</td>
<td>3.2</td>
<td>3.4</td>
</tr>
<tr>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
</tr>
</tbody>
</table>

**ObsErr+AIL_AIRP:** # kx = 16
- u_UprAir | 3.1 | 3.6 | 3.6 | 4.5 | 
- v_UprAir | 3.1 | 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 | 5.7 | 
- 5.7 | 5.7 | 5.7 | 5.7 | 

**ObsErr+AIL_COU3:** # kx = 17
- u_UprAir | 3.1 | 3.6 | 3.6 | 4.5 | 
- v_UprAir | 3.1 | 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 | 5.7 | 
- 5.7 | 5.7 | 5.7 | 5.7 | 

**ObsErr+AIL_LPE:** # kx = 18
- u_UprAir | 4.64 | 4.62 | 4.13 | 3.60 | 
- v_UprAir | 4.64 | 4.62 | 4.13 | 3.60 | 
- 5.80 | 6.00 | 6.50 | 6.50 | 
- 7.00 | 7.00 | 7.00 | 7.00 | 
- 7.00 | 7.00 | 7.00 | 7.00 | 

**ObsErr+CLSTHND:** # kx = 19-27
- u_UprAir | 5.3 | 10.5 | 13.1 | 14.6 | 
- v_UprAir | 5.3 | 10.5 | 13.1 | 14.6 | 
- 15.7 | 17.3 | 18.4 | 19.7 | 
- 24.1 | 24.7 | 24.7 | 28.9 | 
- 29.2 | 31.2 | 33.3 | 36.6 | 
- 38.0 | 40.3 | 42.6 | 42.6 | 
- 42.6 | 42.6 | 42.6 | 42.6 | 

**ObsErr+TSA_NKL:** # kx = 33
- u_UprAir | 4.3 | 8.6 | 10.7 | 12.0 | 
- v_UprAir | 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 | 40.3 | 42.6 | 
- 42.6 | 42.6 | 42.6 | 42.6 | 

**ObsErr+TSA_SNL:** # kx = 34
- u_UprAir | 4.3 | 8.6 | 10.7 | 12.0 | 
- v_UprAir | 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 | 40.3 | 42.6 | 
- 42.6 | 42.6 | 42.6 | 42.6 | 

**ObsErr+TSA_SNL:** # kx = 35
- u_UprAir | 4.3 | 8.6 | 10.7 | 12.0 | 
- v_UprAir | 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 | 40.3 | 42.6 | 
- 42.6 | 42.6 | 42.6 | 42.6 | 

**ObsErr+TSA_SHL:** # kx = 36
- u_UprAir | 4.3 | 8.6 | 10.7 | 12.0 | 
- v_UprAir | 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 | 40.3 | 42.6 | 
- 42.6 | 42.6 | 42.6 | 42.6 | 

**ObsErr+TSC_NKL:** # kx = 37
- u_UprAir | 5.6 | 9.0 | 13.6 | 17.7 | 
- v_UprAir | 5.6 | 9.0 | 13.6 | 17.7 | 
- 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 | 40.3 | 
- 42.6 | 42.6 | 42.6 | 42.6 | 

**ObsErr+TSC_SNL:** # kx = 38
- u_UprAir | 5.6 | 9.0 | 13.6 | 17.7 | 
- v_UprAir | 5.6 | 9.0 | 13.6 | 17.7 | 
- 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 | 40.3 | 
- 42.6 | 42.6 | 42.6 | 42.6 | 

**ObsErr+TSC_SHL:** # kx = 39
- u_UprAir | 4.3 | 8.6 | 10.7 | 12.0 | 
- v_UprAir | 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 | 40.3 | 42.6 |
DAO Office Note 1998-05

ObsErr+TBES-SNW: # xx = 40
R_Uprair.r 4.3 8.6 10.7 12.0 12.8 14.1 15.0 15.6 16.1 16.7 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
vCer_BN.u 3
vCer_BN.c 3
hCer_BN.c 1

ObsErr+TBES-SNW: # xx = 41
R_Uprair.r 4.3 8.6 10.7 12.0 12.8 14.1 15.0 15.6 16.1 16.7 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
vCer_BN.u 3
vCer_BN.c 3
hCer_BN.c 1

ObsErr+TBES-SNW: # xx = 42
R_Uprair.r 4.3 8.6 10.7 12.0 12.8 14.1 15.0 16.1 16.7 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
vCer_BN.u 3
vCer_BN.c 3
hCer_BN.c 1

ObsErr+TBES-SNW: # xx = 43
R_Uprair.r 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 31.6 33.9 34.0
vCer_BN.r 4
vCer_BN.c 4
hCer_BN.c 1

ObsErr+TBES-SNW: # xx = 44
R_Uprair.r 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 31.6 33.9 34.0
vCer_BN.r 4
vCer_BN.c 4
hCer_BN.c 1

ObsErr+GEN-1000: # xx = 87
R_Uprair.r 10.0 18.5 27.2 36.2 40.5 50.5 55.2 59.8 77.7 95.7 136.0 159.0
vCer_BN.r 4
vCer_BN.c 4
hCer_BN.c 1

ObsErr +vCer_BN-1:
GAUSSIAN exp(-1.4d/L)^2
# pres d = L
1000 4800 600
4 4800 600

ObsErr +vCer_BN-1:
# 01Jan96 - Jing G. - This version of vertical correlation function
# is remedied based on GHS/DS-w1.2.18 level table. The original
# table (see 10Jan96 comment below) is not P.D..
# 01Jan96 - Jing G. -
# Title from GEOS_DAS data file "vcorre.data" reads.
# > BACKSIDE VOURS FROM NL, 1M (LIKE VORCERN W/ AIXER VALS >=.1)
#
# press 1000 850 700 500 400 300 200 150 100 70 50 30 10 5 2 1 .4
#
#
# 1000 1.
# 850 .85 1.
# 700 .66 .64 1.
# 600 .48 .38 .40 1.
# 500 .33 .24 .24 .24 1.
# 400 .21 .15 .15 .15 1.
# 300 .09 .07 .07 .14 1.
# 200 .07 .06 .07 .14 1.
# 100 .05 .04 .06 .13 1.
# 70 .04 .03 .03 .08 1.
# 50 .03 .03 .03 .06 1.
# 30 .02 .02 .02 .04 1.
# 20 .02 .02 .02 .03 1.
#
## press 1000 850 700 500 400 300 200 150 100 70 50 30 10 5 2 1 .4
#
#
# 1000 1.
# 850 .85 1.
# 700 .66 .64 1.
# 600 .48 .38 .40 1.
# 500 .33 .24 .24 .24 1.
# 400 .21 .15 .15 .15 1.
# 300 .09 .07 .07 .14 1.
# 200 .07 .06 .07 .14 1.
# 100 .05 .04 .06 .13 1.
# 70 .04 .03 .03 .08 1.
# 50 .03 .03 .03 .06 1.
# 30 .02 .02 .02 .04 1.
# 20 .02 .02 .02 .03 1.
#
###
# Title from GEOS_DAS data file "vcorre.data" reads.
# > VERTICAL CORR OF TIMOS A.B, AND VTRR HOIST ERROR (EIGER)
# > VALS >=.1)
#
# press 1000 850 700 500 400 300 200 150 100 70 50 30 10 5 2 1 .4
#
Appendix D: Complete Covariance Data Life Cycle Diagrams

Complete covariance data life cycle diagrams for the operators $\Gamma^h$, $\Sigma^h$, $\Sigma^\psi$, $\Sigma^\chi$, $C^h$, $C^\psi$, $C^x$, $\Sigma_w$, $\Sigma_c$, $C_w$, and $C_c$ are presented in Figures 53-58 on pages 182 through 187.
DATA LIFE CYCLE FOR $\Gamma^h$

Stage I
Parse Input Data
- `tabl_FEalpha()`
- `rdpars()`

STATE I
VARIABLES REPRESENTING RESOURCES
- `FEalpha_type`
- `FEalpha_desc`
- `FEalpha_npar`
- `FEalpha_nlev`
- `FEalpha_lev(1:FEalpha_nlev)`
- `FEalpha_pars(1:FEalpha_npar,1:FEalpha_nlev)`

STATE II
LOOK-UP TABLES
- Indirect Matrices (IMATs)
  - `Aum_imat(1:nveclev,1:nveclat)`
  - `Aul_imat(1:nveclev,1:nveclat)`
  - `Avm_imat(1:nveclev,1:nveclat)`
  - `Avl_imat(1:nveclev,1:nveclat)`

STATE III
BLOCK MATRIX OPERATORS
- `op_Mx()`: `ktab(i), jtab(i), ktab(i), jtab(i)`
- `op_Pf()`: `ktab(i), jtab(i), ktab(i), jtab(i)`
- `mv_alf(ktab(i), jtab(i))`, `aj_alf()`

Figure 53: Data life cycle for the operator $\Gamma^h$. 

182
Figure 54: Data life cycle for the operator $\Sigma^h$. 
DATA LIFE CYCLE FOR $\Sigma^\Psi$ AND $\Sigma^\chi$

<table>
<thead>
<tr>
<th>Stage I</th>
<th>Parse Input Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tabl_FEsigW()</td>
</tr>
</tbody>
</table>

**Observation Attributes**
- Latitude, level, data type

**Resources**
- FEsigW_rsrc

**State I**

**Variables Representing Resources**
- FEsigW_type
- FEsigW_desc
- FEsigW_npar
- FEsigW_nlev
- FEsigW_lev(1:FEsigW_nlev)
- FEsigW_pars(1:FEsigW_npar,1:FEsigW_nlev)

**State II**

**Look-Up Tables**
- FEsigS_imat(1:nveclev,1:nveclat)
- FEsigV_imat(1:nveclev,1:nveclat)

**State III**

**Block Matrix Operators**
- op_Mx():
  - sigW(1:n_x)
- op_Pf():
  - sigWi(1:n_i)
  - sigWj(1:n_j)
- sigm(1:nsize)

Figure 55: Data life cycle for the operators $\Sigma^\psi$ and $\Sigma^\chi$. 
Figure 56: Data life cycle for the operators $C^h$, $C^\psi$, and $C^\chi$. 

DAO Office Note 1998-05  
Version 1 Dated 12/06/1999
Figure 57: Data life cycle for the operators $\Sigma^0_u$ and $\Sigma^0_c$. 
DATA LIFE CYCLE FOR $C_u^o$ AND $C_c^o$

**Stage I**
Parse Input Data

- set_${\text{EH}}$Cor()
- set_${\text{EV}}$Cor()
- rdpars()
- rdvctbl()

**State I**
Variables Representing Resources

- $\rho_{oc}$
- $\nu_{ou}$ and $\nu_{oc}$
- $n_{hoech}$
- $n_{voec}$
- $n_{hoech}$
- $n_{voec}$
- $\text{type}_{hoech}$
- $\text{type}_{voec}$
- $\text{desc}_{hoech}$
- $\text{desc}_{voec}$
- $\text{lev}_{hoech}$
- $\text{lev}_{voec}$
- $\text{plev}_{hoech}$
- $\text{plev}_{voec}$
- $\text{par}_{hoech}$
- $\text{corr}_{voec}$

**Stage II**
Tabulation

- set_${\text{EO}}$eH$	ext{HH}()$
- intp_5taus()
- intp_hCor()
- intp_vCor()

**State II**
Look-Up Tables

- IMAT pressure levels
- $\text{pveclevs}(1:n_{\text{veclev}})$
- IMAT $\tau$ values calculated by intp_5taus()
- IMAT tables
- $\rho_{oc}$
- $\text{hoech}(1:n_{\text{veclev}},1:n_{\text{HH}+\text{tab}},1:n_{\text{hoech}})$
- $\nu_{ou}$ and $\nu_{oc}$
- $\text{voec}(1:n_{\text{veclev}},1:n_{\text{veclev}},1:n_{\text{voec}})$

**Stage III**
Indirection

- solve4x()
- conjr()
- conjr2()
- slogtab()
- conjr1()
- op_Mx()
- sym_Cxpy()

**State III**
Block Matrix Operators

- IMAT vertical index $k_{\text{tab}}$
- Passed to low-level routines via interfaces
- IMAT $\tau$ indices calculated in low-level routines
- Diagonal blocks stored in upper triangular form as vectors
- $\text{corU}(i)$ $\text{cor0}(i)$
- Off-diagonal blocks stored in 2-D arrays
- $\text{corU}(i,j)$ $\text{cor0}(i,j)$

Figure 58: Data life cycle for the operators $C_u^o$ and $C_c^o$.  
187
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


