AD NUMBER

ADB120256

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; JUL 1987. Other requests shall be referred to Air Force Armament Lab., Eglin AFB, FL. This document contains export-controlled technical data.

AUTHORITY

AFSC/MNOL wright lab ltr dtd 13 Feb 1992

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified

1b. RESTRICTIVE MARKINGS:

2a. SECURITY CLASSIFICATION AUTHORITY:

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE:

3. DISTRIBUTION / AVAILABILITY OF REPORT: Distribution authorized to U.S. Government Agencies and their contractors; CT (over)

4. PERFORMING ORGANIZATION REPORT NUMBER(S):

5. MONITORING ORGANIZATION REPORT NUMBER(S): AFATL-TR-88-18, Vol 9

6a. NAME OF PERFORMING ORGANIZATION: McDonnell Douglas Astronautics Company

6b. OFFICE SYMBOL (If applicable):

7a. NAME OF MONITORING ORGANIZATION: Aeromechanics Division

6c. ADDRESS (City, State, and ZIP Code): P.O. Box 516, St. Louis, MO 63166

7b. ADDRESS (City, State, and ZIP Code): Air Force Armament Laboratory, Eglin AFB, FL 32542-5434

8a. NAME OF FUNDING / SPONSORING ORGANIZATION: STARS Joint Program Office

8b. OFFICE SYMBOL (If applicable):

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F08635-86-C-0025

8c. ADDRESS (City, State, and ZIP Code): Room 3D139 (1211 Fern St), The Pentagon, Washington DC 20301-3081

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
63756D	921C	GZ	57

11. TITLE (Include Security Classification): Common Ada Missile Package (CAMP) Project: Missile Software Parts, Vol 9: Detail Design Documents (Vol 7-12)

12. PERSONAL AUTHOR(S): D. McNicholl, S. Cohen, C. Palmer, et al.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical Note	FROM Sep 85 TO Mar 88	March 1988	422

16. SUPPLEMENTARY NOTATION: SUBJECT TO EXPORT CONTROL LAWS. Availability of this report is specified on verso of front cover. (over)

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Reusable Software, Missile Software, Software Generators, Ada, Parts Composition Systems, Software Parts

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The objective of the CAMP program is to demonstrate the feasibility of reusable Ada software parts in a real-time embedded application area; the domain chosen for the demonstration was that of missile flight software systems. This required that the existence of commonality within that domain be verified (in order to justify the development of parts for that domain), and that software parts be designed which address those areas identified. An associated parts system was developed to support parts usage. Volume 1 of this document is the User's Guide to the CAMP Software parts; Volume 2 is the Version Description Document; Volume 3 is the Software Product Specification; Volumes 4-6 contain the Top-Level Design Document; and, Volumes 7-12 contain the Detail Design Documents.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION: Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL: Christine Anderson

22b. TELEPHONE (Include Area Code): (904) 882-2961

22c. OFFICE SYMBOL: AFATL/FXG

DD Form 1473, JUN 86 — Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

~~this report documents test and evaluation~~; distribution limitation applied March 1988. Other requests for this document must be referred to AFATL/FXG, Eglin AFB, Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

These technical notes accompany the CAMP final report AFATL-TR-85-93 (3 Vols)

AFATL-TR-88-18, Vol 9

SOFTWARE DETAILED DESIGN DOCUMENT

FOR THE

MISSILE SOFTWARE PARTS

OF THE

COMMON ADA MISSILE PACKAGE (CAMP) PROJECT

CONTRACT F08635-86-C-0025

CDRL SEQUENCE NO. C007

30 OCTOBER 1987

AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▮ United States Air Force ▮ Eglin Air Force Base, Florida

3.3.6.2 GENERAL_VECTOR_MATRIX_ALGEBRA (BODY) TLCSC P682 (CATALOG #P197-0)

This part is a package of generic packages and generic functions. The LLCSC's take two different forms. One form defines vector and matrix types, along with general operations on these types. The other form requires that vector and matrix types be provided as generic parameters and performs operations on data objects of different types.

Many of the parts have both an unconstrained and constrained or restricted and unrestricted versions. The constrained/restricted versions of these parts are less flexible in the dimensioning of the input arrays, but require fewer internal calculations.

The generic functions/package which import generic formal array types have been designed to work in conjunction with the data types exported by the generic packages.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
General_Vector_Matrix_Algebra	R058
Vector_Operations_Unconstrained	R061, R062, R063, R104
Vector_Operations_Constrained	R061, R062, R063, R104
Matrix_Operations_Unconstrained	R075, R076, R079, R080, R155, R156
Matrix_Operations_Constrained	R075, R076, R079, R080, R155, R156
Dynamically_Sparse_Matrix_Operations_Unconstrained	R226
Dynamically_Sparse_Matrix_Operations_Constrained	R226
Symmetric_Half_Storage_Matrix_Operations	R211
Symmetric_Full_Storage_Matrix_Operations_Unconstrained	R227
Symmetric_Full_Storage_Matrix_Operations_Constrained	R227
Diagonal_Matrix_Operations	R212
Vector_Scalar_Operations_Unconstrained	R065, R066
Vector_Scalar_Operations_Constrained	R065, R066
Matrix_Scalar_Operations_Unconstrained	R073, R074
Matrix_Scalar_Operations_Constrained	R073, R074
Diagonal_Matrix_Scalar_Operations	R212
Matrix_Vector_Multiply_Unrestricted	R069
Matrix_Vector_Multiply_Restricted	R069
Vector_Matrix_Multiply_Unrestricted	N/A
Vector_Matrix_Multiply_Restricted	N/A
Vector_Vector_Transpose_Multiply_Unrestricted	N/A
Vector_Vector_Transpose_Multiply_Restricted	N/A
Matrix_Matrix_Multiply_Unrestricted	R077
Matrix_Matrix_Multiply_Restricted	R077
Matrix_Matrix_Transpose_Multiply_Unrestricted	N/A
Matrix_Matrix_Transpose_Multiply_Restricted	N/A
Dot_Product_Operation_Unrestricted	R063
Dot_Product_Operation_Restricted	R063
Diagonal_Full_Matrix_Add_Unrestricted	R212
Diagonal_Full_Matrix_Add_Restricted	R212
ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix	N/A
ABA_Trans_Vector_Sq_Matrix	N/A
ABA_Trans_Vector_Scalar	N/A
ABA_Trans_Col_Matrix_Sq_Matrix	N/A
Column_Matrix_Operations	N/A

3.3.6.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.3 INPUT/OUTPUT

None.

3.3.6.2.4 LOCAL DATA

None.

3.3.6.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.6 PROCESSING

The following describes the processing performed by this part:

```
package body General_Vector_Matrix_Algebra is
   package body Vector_Operations_Unconstrained is separate;
   package body Vector_Operations_Constrained is separate;
   package body Matrix_Operations_Unconstrained is separate;
   package body Matrix_Operations_Constrained is separate;
   package body Dynamically_Sparse_Matrix_Operations_Unconstrained is separate;
   package body Dynamically_Sparse_Matrix_Operations_Constrained is separate;
   package body Symmetric_Half_Storage_Matrix_Operations is separate;
   package body Symmetric_Full_Storage_Matrix_Operations_Unconstrained is separate;
   package body Symmetric_Full_Storage_Matrix_Operations_Constrained is separate;
   package body Diagonal_Matrix_Operations is separate;
   package body Vector_Scalar_Operations_Unconstrained is separate;
   package body Vector_Scalar_Operations_Constrained is separate;
   package body Matrix_Scalar_Operations_Unconstrained is separate;
   package body Matrix_Scalar_Operations_Constrained is separate;
   package body Diagonal_Matrix_Scalar_Operations is separate;
   package body Matrix_Vector_Multiply_Unrestricted is separate;
   function Matrix_Vector_Multiply_Restricted
               (Matrix : Input_Matrices;
                Vector : Input_Vectors) return Output_Vectors is separate;
   package body Vector_Vector_Transpose_Multiply_Unrestricted is separate;
   function Vector_Vector_Transpose_Multiply_Restricted
               (Left  : Left_Vectors ;
                Right : Right_Vectors) return Matrices is separate;
```

```
   package body Matrix_Matrix_Multiply_Unrestricted is separate;

   function Matrix_Matrix_Multiply_Restricted
               (Left  : Left_Matrices;
                Right : Right_Matrices) return Output_Matrices is separate;

   package body Matrix_Matrix_Transpose_Multiply_Unrestricted is separate;

   function Matrix_Matrix_Transpose_Multiply_Restricted
               (Left  : Left_Matrices;
                Right : Right_Matrices) return Output_Matrices is separate;

   package body Dot_Product_Operations_Unrestricted is separate;

   function Dot_Product_Operations_Restricted
               (Left  : Left_Vectors;
                Right : Right_Vectors)
               return Result_Elements is separate;

   package body Diagonal_Full_Matrix_Add_Unrestricted is separate;

   function Diagonal_Full_Matrix_Add_Restricted
               (D_Matrix : Diagonal_Matrices;
                F_Matrix : Full_Matrices) return Full_Matrices is separate;

   package body Vector_Matrix_Multiply_Unrestricted is separate;

   function Vector_Matrix_Multiply_Restricted
               (Vector : Input_Vectors;
                Matrix : Input_Matrices) return Output_Vectors is separate;

   package body ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix is separate;

   package body ABA_Trans_Vector_Sq_Matrix is separate;

   package body ABA_Trans_Vector_Scalar is separate;

   package body Column_Matrix_Operations is separate;

end General_Vector_Matrix_Algebra;
```

3.3.6.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.8 LIMITATIONS

None.

3.3.6.2.9 LLCSC DESIGN

3.3.6.2.9.1 VECTOR_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P337-0)

This package contains functions which provide a set of standard vector operations. The operations provided are addition, subtraction, and dot product of like vectors, along with a vector length operation.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.1.1 REQUIREMENTS ALLOCATION

The following table describes the allowing of requirements to this part:

Name	Requirements Allocation
Dot_Product	R063
Vector_Length	R104
"+"	R061
"-"	R062

3.3.6.2.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were defined at the package specification level:

Data types:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_ Squared	floating point type	Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

Subprograms:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared
SqRt	function	Square root function taking an object of type Vector_Elements_Squared and returning an object of type Vector_Elements

3.3.6.2.9.1.4 LOCAL DATA

Data types:

The following table summarizes the types defined in this part's specification:

Name	Range	Description
Vectors	N/A	Unconstrained, one-dimensional array of elements

3.3.6.2.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Operations_Unconstrained is

end Vector_Operations_Unconstrained;
```

3.3.6.2.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.1.8 LIMITATIONS

None.

3.3.6.2.9.1.9 LLCSC DESIGN

None.

3.3.6.2.9.1.10 UNIT DESIGN

3.3.6.2.9.1.10.1 "+" (VECTOR + VECTORS := VECTORS) UNIT DESIGN (CATALOG #P338-0)

This function adds two vectors by adding each of the individual elements in the input vector, returning the resultant vector. All three vectors are of the same type. If the two input vectors do not have the same length, the exception DIMENSION_ERROR is raised. The ranges of the dimensions of the input vectors do not have to be the same.

3.3.6.2.9.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R061.

3.3.6.2.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.1.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	One of the vectors to be added
Right	Vectors	In	Second vector to be added

3.3.6.2.9.1.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Vector being calculated and returned
L_Index	Indices	Index into Left vector
R_Index	Indices	Index into Right vector

3.3.6.2.9.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Vectors;
                 Right : Vectors) return Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Vectors(Left'RANGE);
      L_Index : Indices;
      R_Index : Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --make sure lengths of input vectors are the same
      if Left'LENGTH = Right'LENGTH then

         L_Index := Left'FIRST;
         R_Index := Right'FIRST;

         Process:
            loop

               Answer(L_Index) := Left(L_Index) + Right(R_Index);

               exit Process when L_Index = Left'LAST;

               L_Index := Indices'SUCC(L_Index);
               R_Index := Indices'SUCC(R_Index);

            end loop Process;

      else

--       --dimensions of vectors are incompatible
         raise Dimension_Error;

      end if;

      return Answer;

   end "+";
```

3.3.6.2.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and are defined in the package specification for Vector_Operations_Unconstrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Unconstrained:

Name	Range	Description
Vectors	N/A	Unconstrained, one-dimensional array of elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Type	Description
dimension_error	exception	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.1.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the input vectors are not the same

3.3.6.2.9.1.10.2 "-" (VECTORS - VECTORS := VECTORS) UNIT DESIGN (CATALOG #P339-0)

This part subtracts one vector from another by subtracting the individual elements of each input vector, returning the resultant vector. The dimensions of the two input vectors must have the same length, but are not required to have the same range.

3.3.6.2.9.1.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R062.

3.3.6.2.9.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.1.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	Vector to act as the minuend
Right	Vectors	In	Vector to act as the subtrahend

3.3.6.2.9.1.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Vector being calculated and returned
L_Index	Indices	Index into Left vector
R_Index	Indices	Index into Right vector

3.3.6.2.9.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Vectors;
```

```
                Right : Vectors) return Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Vectors(Left'RANGE);
      L_Index  : Indices;
      R_Index  : Indices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

--    --make sure lengths of the input vectors are the same
      if Left'LENGTH = Right'LENGTH then

         L_Index := Left'FIRST;
         R_Index := Right'FIRST;

         Process:
            loop

               Answer(L_Index) := Left(L_Index) - Right(R_Index);

               exit Process when L_Index = Left'LAST;

               L_Index := Indices'SUCC(L_Index);
               R_Index := Indices'SUCC(R_Index);

            end loop Process;

      else

--       --dimensions of vectors are incompatible
         raise Dimension_Error;

      end if;

      return Answer;

   end "-";
```

3.3.6.2.9.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector_Operations_Unconstrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Unconstrained:

Name	Range	Description
Vectors	N/A	Unconstrained, one-dimensional array of elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Type	Description
dimension_error	exception	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.1.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the input vectors are not the same

3.3.6.2.9.1.10.3 VECTOR_LENGTH UNIT DESIGN (CATALOG #2340-0)

This function calculates the length of a vector, returning the result. The length of a vector is defined as:

```
a := Sqrt(sum b(i)**2)
```

3.3.6.2.9.1.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R104.

3.3.6.2.9.1.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.1.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Vectors	In	Vector for which a length is desired

3.3.6.2.9.1.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Temp	Vector_Elements_Squared	Used for intermediate calculations

3.3.6.2.9.1.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   function Vector_Length (Input : Vectors) return Vector_Elements is

--    ----------------------
--    --declaration section-
--    ----------------------

      Temp   : Vector_Elements_Squared;

-- ------------------------------
-- --begin function Vector_Length
-- ------------------------------
```

```
   begin

      Temp := 0.0;

      Process:
         for Index in Input'RANGE loop
            Temp := Temp +
                    Input(Index) * Input(Index);
         end loop Process;

      return SqRt(Temp);

   end Vector_Length;
```

3.3.6.2.9.1.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector_Operations_Unconstrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_ Squared	floating point type	Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Unconstrained:

Name	Range	Description
Vectors	N/A	Unconstrained, one-dimensional array of elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Type	Mode	Description
Left	Vectors	In	First vector to be used in the dot product operation
Right	Vectors	In	Second vector to be used in the dot product operation

3.3.6.2.9.1.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vector_Elements_Squared	Result of the dot product operation
L_Index	Indices	Index into Left vector
R_Index	Indices	Index into Right vector

3.3.6.2.9.1.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.1.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function Dot_Product (Left  : Vectors;
                         Right : Vectors) return Vector_Elements_Squared is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Vector_Elements_Squared;
      L_Index  : Indices;
      R_Index  : Indices;

-- ---------------------------
-- --begin function Dot_Product
-- ---------------------------

   begin

--    --make sure lengths of the input vectors are the same
      if Left'LENGTH = Right'LENGTH then

         Answer := 0.0;
         L_Index := Left'FIRST;
         R_Index := Right'FIRST;
```

```
         Process:
            loop

               Answer := Answer + Left(L_Index) * Right(R_Index);

               exit Process when L_Index = Left'LAST;

               L_Index := Indices'SUCC(L_Index);
               R_Index := Indices'SUCC(R_Index);

            end loop Process;

      else

--       --dimensions of vectors are incompatible
         raise Dimension_Error;

      end if;

      return Answer;

   end Dot_Product;
```

3.3.6.2.9.1.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector_Operations_Unconstrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_ Squared	floating point type	Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Unconstrained:

Name	Range	Description
Vectors	N/A	Unconstrained, one-dimensional array of elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Type	Description
dimension_error	exception	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

Subprograms:

The following table summarizes the generic subroutines available to this part and defined at the package specification level for Vector_Operations:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared

3.3.6.2.9.1.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the two input vectors are not the same

3.3.6.2.9.2 MATRIX_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P347-0)

This package contains subroutines which provide a set of standard operations on matrices of like types.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.2.1 REQUIREMENTS ALLOCATION

This following illustrates the allocation of requirements to the units in this package.

Name	Requirements Allocation
"+" (matrices + matrices)	R079
"-" (matrices - matrices)	R080
"+" (matrices + elements)	R075
"-" (matrices - elements)	R076
Set_to_Identity_Matrix	R155
Set_to_Zero_Matrix	R156
"*"	R077

3.3.6.2.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level:

Data types:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

3.3.6.2.9.2.4 LOCAL DATA

Data types:

The following data type was previously defined at the package specification level:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Operations_Unconstrained is

end Matrix_Operations_Unconstrained;
```

3.3.6.2.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.2.8 LIMITATIONS

None.

3.3.6.2.9.2.9 LLCSC DESIGN

None.

3.3.6.2.9.2.10 UNIT DESIGN

3.3.6.2.9.2.10.1 "+" (MATRICES + MATRICES := MATRICES) UNIT DESIGN (CATALOG #P348-0)

This function adds two matrices by adding the individual elements of each input matrix, returning the resultant matrix. The lengths of the first dimensions of the input matrices must be equal, as must be the lengths of the second dimensions. None of the ranges for the dimensions have to be the same.

3.3.6.2.9.2.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R079.

3.3.6.2.9.2.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	First matrix to be added
Right	Matrices	In	Second matrix to be added

3.3.6.2.9.2.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Result of adding the two input matrices
L_Col	Col_Indices	Left column index
L_Row	Row_Indices	Left row index
R_Col	Col_Indices	Right column index
R_Row	Row_Indices	Right row index

3.3.6.2.9.2.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Matrices(Left'RANGE(1), Left'RANGE(2));
      L_Col    : Col_Indices;
      L_Row    : Row_Indices;
      R_Col    : Col_Indices;
      R_Row    : Row_Indices;
```

```
-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --make sure the dimensions of the matrices are compatible
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then

         L_Row := Left'FIRST(1);
         R_Row := Right'FIRST(1);
         Row_Loop:
            loop

               L_Col := Left'FIRST(2);
               R_Col := Right'FIRST(2);
               Col_Loop:
                  loop

                     Answer(L_Row, L_Col) := Left(L_Row, L_Col) +
                                             Right(R_Row, R_Col);

                     exit Col_Loop when L_Col = Left'LAST(2);
                     L_Col := Col_Indices'SUCC(L_Col);
                     R_Col := Col_Indices'SUCC(R_Col);

                  end loop Col_Loop;

               exit Row_Loop when L_Row = Left'LAST(1);
               L_Row := Row_Indices'SUCC(L_Row);
               R_Row := Row_Indices'SUCC(R_Row);

            end loop Row_Loop;

      else

--       --input matrices have incompatible dimensions
         raise Dimension_Error;

      end if;

      return Answer;

   end "+";
```

3.3.6.2.9.2.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.2.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the respective lengths of the first and second dimensions of the input matrices are not equal

3.3.6.2.9.2.10.2 "-" (MATRICES - MATRICES := MATRICES) UNIT DESIGN (CATALOG #P349-0)

This function subtracts one matrix from another by subtracting the individual elements of the input matrices, returning the resultant matrix. The lengths of

the first dimensions of the input matrices must be equal, as must be the lengths of the second dimensions. None of the ranges for the dimensions have to be the same.

3.3.6.2.9.2.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R076.

3.3.6.2.9.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Matrix to act as the minuend
Right	Matrices	In	Matrix to be used as the subtrahend

3.3.6.2.9.2.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Result of adding the two input matrices
L_Col	Col_Indices	Left column index
L_Row	Row_Indices	Left row index
R_Col	Col_Indices	Right column index
R_Row	Row_Indices	Right row index

3.3.6.2.9.2.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is
```

```
--     ----------------------
--     --declaration section-
--     ----------------------

       Answer    : Matrices(Left'RANGE(1), Left'RANGE(2));
       L_Col     : Col_Indices;
       L_Row     : Row_Indices;
       R_Col     : Col_Indices;
       R_Row     : Row_Indices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

--     --make sure matrix dimensions are compatible
       if Left'LENGTH(1) = Right'LENGTH(1) and
          Left'LENGTH(2) = Right'LENGTH(2) then

          L_Row := Left'FIRST(1);
          R_Row := Right'FIRST(1);
          Row_Loop:
             loop

                L_Col := left'FIRST(2);
                R_Col := Right'FIRST(2);
                Col_Loop:
                   loop

                      answer(L_Row, L_Col) := Left(L_Row, L_Col) -
                                              Right(R_Row, R_Col);

                      exit Col_Loop when L_Col = Left'LAST(2);
                      L_Col := Col_Indices'SUCC(L_Col);
                      R_Col := Col_Indices'SUCC(R_Col);

                   end loop Col_Loop;

                exit Row_Loop when L_Row = Left'LAST(1);
                L_Row := Row_Indices'SUCC(L_Row);
                R_Row := Row_Indices'SUCC(R_Row);

             end loop Row_Loop;

       else

--        --input matrices have incompatible dimensions
          raise Dimension_Error;

       end if;

       return Answer;

   end "-";
```

3.3.6.2.9.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification for Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.2.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the respective lengths of the first and second dimensions of the input matrices are not the same

3.3.6.2.9.2.10.3 "+" (MATRICES + ELEMENTS := MATRICES) UNIT DESIGN (CATALOG #P350-0)

This function calculates a scaled matrix by adding a scale factor to each element of an input matrix, returning the resultant matrix.

3.3.6.2.9.2.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R075.

3.3.6.2.9.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be scaled
Addend	Elements	In	Scale factor

3.3.6.2.9.2.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Scaled matrix

3.3.6.2.9.2.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Matrix : Matrices;
                 Addend : Elements) return Matrices is

--    ---------------------
```

```
--     --declaration section-
--     ----------------------

       Answer : Matrices(Matrix'RANGE(1), Matrix'RANGE(2));

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Row_Loop:
         for Row in Matrix'RANGE(1) loop
            Col_Loop:
               for Col in Matrix'RANGE(2) loop
                  Answer(Row, Col) := Matrix(Row, Col) + Addend;
               end loop Col_Loop;
         end loop Row_Loop;

      return Answer;

   end "+";
```

3.3.6.2.9.2.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.2.10.3.8 LIMITATIONS

None.

3.3.6.2.9.2.10.4 "-" (MATRICES - ELEMENTS := MATRICES) UNIT DESIGN (CATALOG #P351-0)

This function calculates a scaled matrix by subtracting a scale factor from each element of an input matrix, returning the resultant matrix.

3.3.6.2.9.2.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R076.

3.3.6.2.9.2.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be scaled
Subtrahend	Elements	In	Scale factor

3.3.6.2.9.2.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Scaled matrix

3.3.6.2.9.2.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Matrix     : Matrices;
                 Subtrahend : Elements) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices(Matrix'RANGE(1), Matrix'RANGE(2));

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Row_Loop:
         for Row in Matrix'RANGE(1) loop
            Col_Loop:
               for Col in Matrix'RANGE(2) loop
                  Answer(Row, Col) := Matrix(Row, Col) - Subtrahend;
               end loop Col_Loop;
         end loop Row_Loop;

      return Answer;

   end "-";
```

3.3.6.2.9.2.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.2.10.4.8 LIMITATIONS

None.

3.3.6.2.9.2.10.5 SET_TO_IDENTITY_MATRIX UNIT DESIGN (CATALOG #P352-0)

This procedure turns an input matrix into an identity matrix. An identity matrix is one in which the diagonal elements equal 1.0 and all other elements equal 0.0. The input matrix must be a square matrix, but the ranges of the individual dimensions do not have to be the same.

3.3.6.2.9.2.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R155.

3.3.6.2.9.2.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	Out	Matrix to be made into an identity matrix

3.3.6.2.9.2.10.5.4 LOCAL DATA

Data objects:

The following data objects are maintained local to this part.

Name	Type	Description
Col_Marker	Col_Indices	Index into second dimension of matrix
Row	Row_Indices	Index into first dimension of matrix

3.3.6.2.9.2.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--    ---------------------
--    --declaration section
--    ---------------------

      Col_Marker : Col_Indices;
      Row        : Row_Indices;

-- ---------------------------------------
-- --begin function Set_To_Identity_Matrix
-- ---------------------------------------

   begin

--    --make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

         Matrix := (others => (others => 0.0));

         Row        := Matrix'FIRST(1);
         Col_Marker := Matrix'FIRST(2);
         Row_Loop:
            loop

--             --set diagonal element equal to 1
               Matrix(Row, Col_Marker) := 1.0;

               exit Row_Loop when Row = Matrix'LAST(1);
               Row        := Row_Indices'SUCC(Row);
               Col_Marker := Col_Indices'SUCC(Col_Marker);

            end loop Row_Loop;

      else

--       --do not have a square matrix
         raise Dimension_Error;

      end if;
```

```
   end Set_To_Identity_Matrix;
```

3.3.6.2.9.2.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.2.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	Description
Dimension_Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.2.10.6 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P353-0)

This procedure zeros out all elements of an input matrix.

3.3.6.2.9.2.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R156.

3.3.6.2.9.2.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	Out	Matrix to be zeroed out

3.3.6.2.9.2.10.6.4 LOCAL DATA

None.

3.3.6.2.9.2.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
begin
   Matrix := (others => (others => 0.0));
end Set_To_Zero_Matrix;
```

3.3.6.2.9.2.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification for Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.2.10.6.8 LIMITATIONS

None.

3.3.6.2.9.2.10.7 "*" (MATRICES * MATRICES => MATRICES) UNIT DESIGN (CATALOG #P354-0)

This function multiplies an m x n matrix by an n x p matrix, returning and m x p matrix. The type of elements in each of the three matrices is the same.

The values in the result matrix are defined as:

```
a(m,p) := b(m,n) * c(n,p)
```

3.3.6.2.9.2.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.3.6.2.9.2.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	m x n matrix to act as multiplicand
Right	Matrices	In	n x p matrix to act as multiplier

3.3.6.2.9.2.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of multiplying two input matrices
M	Row_Indices	N/A	Index into rows of left and answer matrices
N_Left	Col_Indices	N/A	Index into columns of left matrix
N_Right	Row_Indices	N/A	Index into rows of right matrix
P	Col_Indices	N/A	Index into colums of left and answer matrices

3.3.6.2.9.2.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    --------------------
--    --declaration section
--    --------------------

      Answer  : Matrices(Left'RANGE(1), Right'RANGE(2));
      M       : Row_Indices;
      N_Left  : Col_Indices;
```

```
      N_Right : Row_Indices;
      P       : Col_Indices;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

--    --make sure dimensions are compatible
      if Left'LENGTH(2) = Right'LENGTH(1) then

         M := Left'FIRST(1);
         M_Loop:
            loop

               P := Right'FIRST(2);
               P_Loop:
                  loop

                     Answer(M,P) := 0.0;

                     N_Left  := Left'FIRST(2);
                     N_Right := Right'FIRST(1);
                     N_Loop:
                        loop

                           Answer(M,P) := Answer(M,P) +
                                          Left(M,N_Left) * Right(N_Right,P);

                           exit N_Loop when N_Left = Left'LAST(2);
                           N_Left  := Col_Indices'SUCC(N_Left);
                           N_Right := Row_Indices'SUCC(N_Right);

                        end loop N_Loop;

                     exit P_Loop when P = Right'LAST(2);
                     P := Col_Indices'SUCC(P);

                  end loop P_Loop;

               exit M_Loop when M = Left'LAST(1);
               M := Row_Indices'SUCC(M);

            end loop M_Loop;

      else

--       --dimensions are incompatible
         raise Dimension_Error;

      end if;

      return Answer;

   end "*";
```

3.3.6.2.9.2.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Matrix_Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification for Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

Subprograms:

The following table summarizes the generic subroutines available to this part and defined at the package specification level of Matrix_Operations_-Unconstrained:

Name	Type	Description
"*"	function	Operator to define the operation Elements * Elements => Elements

3.3.6.2.9.2.10.7.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the inner dimensions of the input matrices do not have the same length

3.3.6.2.9.3 DYNAMICALLY_SPARSE_MATRIX_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P362-0)

This package defines a dynamically sparse matrix and operations on it. All elements of the matrix are stored, but most of the elements are expected to be 0. Which elements are zero does not have to remain the same. See decomposition section for the operations provided.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described at the package specification level:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

3.3.6.2.9.3.4 LOCAL DATA

Data types:

The following data types were previously defined at the package specification level:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Dynamically_Sparse_Matrix_Operations_Unconstrained is

end Dynamically_Sparse_Matrix_Operations_Unconstrained;
```

3.3.6.2.9.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.3.8 LIMITATIONS

None.

3.3.6.2.9.3.9 LLCSC DESIGN

None.

3.3.6.2.9.3.10 UNIT DESIGN

3.3.6.2.9.3.10.1 SET_TO_IDENTITY_MATRIX UNIT DESIGN (CATALOG #P363-0)

This procedure sets a square input matrix to an identity matrix. An identity matrix is one where the diagonal elements all equal 1.0, with the remaining elements equaling 0.0.

3.3.6.2.9.3.10.1.1 REQUIREMENTS ALLOCATION

See main header.

3.3.6.2.9.3.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix being made into an identity matrix

3.3.6.2.9.3.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Col_Marker	Col_Indices	Index into second dimension of input matrix
Row	Row_Indices	Index into first dimension of input matrix

3.3.6.2.9.3.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Set_To_Identity_Matrix (Matrix : out Matrices) is
```

```
--    ---------------------
--    --declaration section
--    ---------------------

      Col_Marker : Col_Indices;
      Row        : Row_Indices;

-- ---------------------------------------
-- --begin procedure Set_to_Identity_Matrix
-- ---------------------------------------

   begin

--    --make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

         Matrix := (others => (others => 0.0));

         Row        := Matrix'FIRST(1);
         Col_Marker := Matrix'FIRST(2);
         Row_Loop:
            loop

--             --set diagonal element equal to 1.0
               Matrix(Row, Col_Marker) := 1.0;

               exit Row_Loop when Row = Matrix'LAST(1);
               Row        := Row_Indices'SUCC(Row);
               Col_Marker := Col_Indices'SUCC(Col_Marker);

            end loop Row_Loop;

      else

         raise Dimension_Error;

      end if;

   end Set_to_Identity_Matrix;
```

3.3.6.2.9.3.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
Dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.3.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.3.10.2 SET_TO_ZERO_MATRIX UNIT DESIGN (CATALOG #P364-0)

This procedure sets all elements of an input matrix to zero.

3.3.6.2.9.3.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be zeroed out

3.3.6.2.9.3.10.2.4 LOCAL DATA

None.

3.3.6.2.9.3.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

begin

   Matrix := (others => (others => 0.0));

end Set_to_Zero_Matrix;
```

3.3.6.2.9.3.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_Sparse_Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.3.10.2.8 LIMITATIONS

None.

3.3.6.2.9.3.10.3 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P365-0)

This function takes a square input matrix and adds it to an identity matrix by adding 1.0 to all diagonal elements of the input matrix.

3.3.6.2.9.3.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Matrix to which is added an identity matrix

3.3.6.2.9.3.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding an identity matrix to the input matrix
Col_Marker	Col_Indices	N/A	Column index
Row	Row_Indices	N/A	Row index

3.3.6.2.9.3.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   function Add_to_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer     : Matrices(Input'RANGE(1),Input'RANGE(2));
      Col_Marker : Col_Indices;
      Row        : Row_Indices;

-- ---------------------------------
-- --begin procedure Add_to_Identity
-- ---------------------------------

   begin

--    --make sure input is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then

         Answer := Input;

--       --add "identity" values to diagonal elements
         Row        := Input'FIRST(1);
         Col_Marker := Input'FIRST(2);
         Row_Loop:
            loop

               if Answer(Row, Col_Marker) /= 0.0 then
                  Answer(Row, Col_Marker) := Answer(Row, Col_Marker) + 1.0;
               else
                  Answer(Row, Col_Marker) := 1.0;
               end if;
```

```
            exit Row_Loop when Row = Input'LAST(1);
            Row        := Row_Indices'SUCC(Row);
            Col_Marker := Col_Indices'SUCC(Col_Marker);

         end loop Row_Loop;

      else

         raise Dimension_Error;

      end if;

      return Answer;

   end Add_to_Identity;
```

3.3.6.2.9.3.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.3.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.3.10.4 SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P366-0)

This function subtracts a square input matrix from an identity matrix by negating all elements of an input matrix and then adding 1.0 to the elements on the diagonal.

3.3.6.2.9.3.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Square matrix to be subtracted from an identity matrix

3.3.6.2.9.3.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of subtracting input matrix from an identity matrix
Col_Marker	Col_Indices	N/A	Column index
Row	Row_Indices	N/A	Row index

3.3.6.2.9.3.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function Subtract_from_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer     : Matrices(Input'RANGE(1),Input'RANGE(2));
      Col_Marker : Col_Indices;
      Row        : Row_Indices;

-- ----------------------------------
-- --begin procedure Subtract_From_Identity
-- ----------------------------------

   begin

--    --make sure input is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then

         Row        := Input'FIRST(1);
         Col_Marker := Input'FIRST(2);
         Row_Loop:
            loop

               Col_Loop:
                  for Col in Input'RANGE(2) loop
                     if Input(Row,Col) /= 0.0 then
                        Answer(Row,Col) := - Input(Row,Col);
                     else
                        Answer(Row,Col) := 0.0;
                     end if;
                  end loop Col_Loop;

               if Answer(Row, Col_Marker) /= 0.0 then
                  Answer(Row, Col_Marker) := Answer(Row, Col_Marker) + 1.0;
               else
```

```
                Answer(Row, Col_Marker) := 1.0;
             end if;

             exit Row_Loop when Row = Input'LAST(1);
             Row        := Row_Indices'SUCC(Row);
             Col_Marker := Col_Indices'SUCC(Col_Marker);

          end loop Row_Loop;

       else

          raise Dimension_Error;

       end if;

       return Answer;

    end Subtract_From_Identity;
```

3.3.6.2.9.3.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.3.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.3.10.5 "+" UNIT DESIGN (CATALOG #P367-0)

This function adds two sparse m x n matrices, by adding the individual elements of the input matrices taking advantage of the fact that most of the elements of both matrices equal 0.

3.3.6.2.9.3.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Sparse matrix to be added
Right	Matrices	In	Sparse matrix to be added

3.3.6.2.9.3.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding two input matrices
L_Col	Col_Indices	N/A	Column index into left matrix
L_Row	Row_Indices	N/A	Row index into left matrix
R_Col	Col_Indices	N/A	Column index into right matrix
R_Row	Row_Indices	N/A	Row index into right matrix

3.3.6.2.9.3.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer : Matrices(Left'RANGE(1), Left'RANGE(2));
       L_Col  : Col_Indices;
       L_Row  : Row_Indices;
       R_Col  : Col_Indices;
       R_Row  : Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--     --make sure have compatible dimensions
       if Left'LENGTH(1) = Right'LENGTH(1) and then
          Left'LENGTH(2) = Right'LENGTH(2) then

          L_Row := Left'FIRST(1);
          R_Row := Right'FIRST(1);
          Row_Loop:
             loop

                L_Col := Left'FIRST(2);
                R_Col := Right'FIRST(2);
                Col_Loop:
```

```
                loop

                   if Left(L_Row, L_Col) = 0.0 then
                      if Right(R_Row, R_Col) = 0.0 then
                         Answer(L_Row, L_Col) := 0.0;
                      else
                         Answer(L_Row, L_Col) := Right(R_Row, R_Col);
                      end if;
                   elsif Right(R_Row, R_Col) = 0.0 then
                      Answer(L_Row, L_Col) := Left(L_Row, L_Col);
                   else
                      Answer(L_Row, L_Col) := Left(L_Row, L_Col) +
                                              Right(R_Row, R_Col);
                   end if;

                   exit Col_Loop when L_Col = Left'LAST(2);
                   L_Col := Col_Indices'SUCC(L_Col);
                   R_Col := Col_Indices'SUCC(R_Col);

                end loop Col_Loop;

             exit Row_Loop when L_Row = Left'LAST(1);
             L_Row := Row_Indices'SUCC(L_Row);
             R_Row := Row_Indices'SUCC(R_Row);

          end loop Row_Loop;

       else

          raise Dimension_Error;

       end if;

       return Answer;

    end "+";
```

3.3.6.2.9.3.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.3.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if both matrices are not m x n matrices

3.3.6.2.9.3.10.6 "-" UNIT DESIGN (CATALOG #P368-0)

This function subtracts two sparse m x n matrices by subtracting the individual elements of the input matrices, taking advantage of the fact that most of the elements of both matrices equal 0.

3 3.6.2.9.3.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Sparse matrix to be treated as the minuend
Right	Matrices	In	Sparse matrix to be treated as the subtrahend

3.3.6.2.9.3.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of subtracting two input matrices
L_Col	Col_Indices	N/A	Column index into left matrix
L_Row	Row_Indices	N/A	Row index into left matrix
R_Col	Col_Indices	N/A	Column index into right matrix
R_Row	Row_Indices	N/A	Row index into right matrix

3.3.6.2.9.3.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    --------------------
--    --declaration section
--    --------------------
```

```
      Answer : Matrices(Left'RANGE(1), Left'RANGE(2));
      L_Col  : Col_Indices;
      L_Row  : Row_Indices;
      R_Col  : Col_Indices;
      R_Row  : Row_Indices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

--    --make sure have compatible dimensions
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then

         L_Row := Left'FIRST(1);
         R_Row := Right'FIRST(1);
         Row_Loop:
            loop

               L_Col := Left'FIRST(2);
               R_Col := Right'FIRST(2);
               Col_Loop:
                  loop

                     if Left(L_Row, L_Col) = 0.0 then
                        if Right(R_Row, R_Col) = 0.0 then
                           Answer(L_Row, L_Col) := 0.0;
                        else
                           Answer(L_Row, L_Col) := - Right(R_Row, R_Col);
                        end if;
                     elsif Right(R_Row, R_Col) = 0.0 then
                        Answer(L_Row, L_Col) := Left(L_Row, L_Col);
                     else
                        Answer(L_Row, L_Col) := Left(L_Row, L_Col) -
                                                Right(R_Row, R_Col);
                     end if;

                     exit Col_Loop when L_Col = Left'LAST(2);
                     L_Col := Col_Indices'SUCC(L_Col);
                     R_Col := Col_Indices'SUCC(R_Col);

                  end loop Col_Loop;

               exit Row_Loop when L_Row = Left'LAST(1);
               L_Row := Row_Indices'SUCC(L_Row);
               R_Row := Row_Indices'SUCC(R_Row);

            end loop Row_Loop;

      else

         raise Dimension_Error;

      end if;
```

```
      return Answer;

   end "-";
```

3.3.6.2.9.3.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.3.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if both matrices are not m x n matrices

3.3.6.2.9.4 SYMMETRIC_HALF_STORAGE_MATRIX_OPERATIONS PACKAGE DESIGN (CATALOG #P376-0)

This package defines a symmetric half storage matrix and provides operations on it. For the operations provided, see the decomposition section. The bottom half of the matrix will be stored in row-major order.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.2 LOCAL ENTITIES DESIGN

Subprograms:

The following table describes the subprograms local to this part:

Name	Type	Description
Swap_Row	function	Takes a column as input, and returns the corresponding row entry
Swap_Col	function	Takes a row as input, and returns the corresponding column entry

This package contains code which is executed when the package is elaborated. This code first checks to make sure a square matrix has been instantiated. If not, a Dimension_Error exception is raised. If a square matrix has been instantiated, this the code initializes the Row_Marker, Local_Identity_Matrix, and Col_Offset arrays.

3.3.6.2.9.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in this part's package specification.

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored
Col_Slices	array	Data type defining a column slice of a matrix
Row_Slices	array	Data type defining a row slice of a matrix

3.3.6.2.9.4.4 LOCAL DATA

Data types:

The following table describes the data types previously defined in this part's package specification:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

The following table describes the data types defined local to this package.

Name	Type	Description
Col_Index_Arrays	array	Array of integers indexed by Indices used to set up column markers into Matrices array
Row_Index_Arrays	array	Array of integers indexed by Indices used to set up row markers into Matrices array

Data objects:

The following table describes the data objects defined in this part's package specification:

Name	Type	Value	Description
Entry_Count	Positive	---	Number of stored values from the half-storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2

The following table describes the data objects maintained by this package:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins
Local_Identity_Matrix	Matrices	Pre-initialized identity matrix
Local_Zero_Matrix	Matrices	Pre-initialized zero matrix

Note: The following scheme is used to access an element:

```
Full_Storage(i,j) <==>
Half_Storage(Row_Marker(i) + Col_Offset(j));
```

The following data objects are contained in a declare block located at the end of this package body:

Name	Type	Value	Description
Count	Natural	N/A	Counts the position of the row index; goes from 0 to the number of rows - 1
Offset	Natural	N/A	Offset of the current column index from the first column index
Row_Starting_Point	Natural	N/A	Where in the diagonal matrix the current row starts

3.3.6.2.9.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)

```
package body Symmetric_Half_Storage_Matrix_Operations is

-- --------------------
-- --local declarations
-- --------------------

   type Col_Index_Arrays is array(Col_Indices) of NATURAL;
   type Row_Index_Arrays is array(Row_Indices) of NATURAL;

   Col_Offset : Col_Index_Arrays;
   Row_Marker : Row_Index_Arrays;

-- --this object is initially only zeroed out; the 1.0 values will be assigned
-- --to the diagonal elements during package initialization
   Local_Identity_Matrix : Matrices := (others => 0.0);

   Local_Zero_Matrix     : constant Matrices := (others => 0.0);


-----------------------------------------------
--begin processing for Symmetric_Half_Storage_
--Matrix_Operations package body
-----------------------------------------------

begin

   Init_Block:
      declare

         Count              : NATURAL;
         Offset             : NATURAL;
         Row_Starting_Point : NATURAL;

      begin

--       --make sure lengths of row and col indices are the same
         if Row_Slices'LENGTH /= Col_Slices'LENGTH then

            raise Dimension_Error;

         else

--          ---------------------------------------------------------------
--          --initialize row marker identity matrix arrays;
--          --all diagonal elements, except for the last one, which require
--          --  a value of 1 for the identity matrix are located one entry
--          --  before the starting location of the next row
--          ---------------------------------------------------------------

--          --handle first row marker entry to simplify initialization of
--          --the identity matrix --(NOTE: count implicitly equals 0)
            Row_Marker(Row_Indices'FIRST) := 1;

            Count := 1;
            Row_Marker_and_Identity_Matrix_Init_Loop:
               for Index in Row_Indices'SUCC(Row_Indices'FIRST) ..
                            Row_Indices'LAST loop
```

```
                    Row_Starting_Point := (Count * (Count+1) / 2) + 1;

                    Row_Marker(Index) := Row_Starting_Point;
                    Local_Identity_Matrix(Row_Starting_Point-1) := 1.0;

                    Count := Count + 1;

                 end loop Row_Marker_and_Identity_Matrix_Init_Loop;

--            --initialize last diagonal element
              Local_Identity_Matrix(Entry_Count) := 1.0;

--            -------------------------------
--            --initialize column offset array
--            -------------------------------

              Offset := 0;
              Col_Marker_Init_Loop:
                 for Index in Col_Indices loop
                    Col_Offset(Index) := Offset;
                    Offset := Offset + 1;
                 end loop Col_Marker_Init_Loop;

           end if;

        end Init_Block;

end Symmetric_Half_Storage_Matrix_Operations;
```

3.3.6.2.9.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if an attempt is made to instantiated other than a square matrix

3.3.6.2.9.4.9 LLCSC DESIGN

None.

3.3.6.2.9.4.10 UNIT DESIGN

3.3.6.2.9.4.10.1 SWAP_COL UNIT DESIGN

This function takes a row index is input, and returns the corresponding column index.

3.3.6.2.9.4.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Row	Row_Indices	In	Row to be converted to a column entry

3.3.6.2.9.4.10.1.4 LOCAL DATA

None.

3.3.6.2.9.4.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Swap_Col (Row : Row_Indices) return Col_Indices is
begin
   return Col_Indices'VAL(Row_Indices'POS(Row) -
                          Row_Indices'POS(Row_Indices'FIRST) +
                          Col_Indices'POS(Col_Indices'FIRST));
end Swap_Col;
```

3.3.6.2.9.4.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

3.3.6.2.9.4.10.1.8 LIMITATIONS

None.

3.3.6.2.9.4.10.2 SWAP_ROW UNIT DESIGN

This function takes a column index is input, and returns the corresponding row index.

3.3.6.2.9.4.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Col	Col_Indices	In	Column to be converted to a row entry

3.3.6.2.9.4.10.2.4 LOCAL DATA

None.

3.3.6.2.9.4.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function Swap_Row (Col : Col_Indices) return Row_Indices is
begin
   return Row_Indices'VAL(Col_Indices'POS(Col) -
                          Col_Indices'POS(Col_Indices'FIRST) +
                          Row_Indices'POS(Row_Indices'FIRST));
end Swap_Row;
```

3.3.6.2.9.4.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

3.3.6.2.9.4.10.2.8 LIMITATIONS

None.

3.3.6.2.9.4.10.3 INITIALIZE UNIT DESIGN (CATALOG #P379-0)

This function allows the half storage matrix to be initialized a row at a time. Even through an entire row of values is sent in, only those applicable to a given row will be referenced (i.e., 1 value for row 1, 2 values for row 2, etc.).

3.3.6.2.9.4.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Row_Slice	Slices	In	Row of values to be placed in the half-storage matrix
Row	Row Indices	In	Indicates which row of values this is
Matrix	Matrices	Out	Half-storage matrix to be initialized

3.3.6.2.9.4.10.3.4 LOCAL DATA

None.

3.3.6.2.9.4.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Initialize (Row_Slice : in     Row_Slices;
                         Row       : in     Row_Indices;
                         Matrix    :    out Matrices) is
```

```
--    ---------------------
--    --declaration section
--    ---------------------

      Index     : Col_Indices;
      Marker    : POSITIVE;
      Stop_Here : POSITIVE;

-- ----------------------------
-- --begin procedure Initialize
-- ----------------------------

   begin

      Index     := Col_Indices'FIRST;
      Marker    := Row_Marker(Row);
      Stop_Here := Marker + Col_Offset(Swap_Col(Row));
      Process:
         loop

            Matrix(Marker) := Row_Slice(Index);

            exit Process when Marker = Stop_Here;
            Index  := Col_Indices'SUCC(Index);
            Marker := Marker + 1;

         end loop Process;

   end Initialize;
```

3.3.6.2.9.4.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored
Col_Slices	array	Data type defining a column slice of a matrix
Row_Slices	array	Data type defining a row slice of a matrix

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Swap_Row	function	Takes a column as input, and returns the corresponding row entry
Swap_Col	function	Takes a row as input, and returns the corresponding column entry

3.3.6.2.9.4.10.3.8 LIMITATIONS

None.

3.3.6.2.9.4.10.4 IDENTITY_MATRIX UNIT DESIGN (CATALOG #P380-0)

This function returns an identity matrix. An identity matrix is one where all elements equal 0.0 except the diagonal elements which equal 1.0.

3.3.6.2.9.4.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.4.3 INPUT/OUTPUT

None.

3.3.6.2.9.4.10.4.4 LOCAL DATA

None.

3.3.6.2.9.4.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
function Identity_Matrix return Matrices is
begin
   return Local_Identity_Matrix;
end Identity_Matrix;
```

3.3.6.2.9.4.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Local_Identity_Matrix	Matrices	Pre-initialized identity matrix

3.3.6.2.9.4.10.4.8 LIMITATIONS

None.

3.3.6.2.9.4.10.5 ZERO_MATRIX UNIT DESIGN (CATALOG #P381-0)

This function returns a zeroed out half-storage matrix.

3.3.6.2.9.4.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.5.3 INPUT/OUTPUT

None.

3.3.6.2.9.4.10.5.4 LOCAL DATA

None.

3.3.6.2.9.4.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
function Zero_Matrix return Matrices is

begin

   return Local_Zero_Matrix;

end Zero_Matrix;
```

3.3.6.2.9.4.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Local_Zero_Matrix	Matrices	Pre-initialized zero matrix

3.3.6.2.9.4.10.5.8 LIMITATIONS

None.

3.3.6.2.9.4.10.6 CHANGE_ELEMENT UNIT DESIGN (CATALOG #P382-0)

This procedure changes a single element in the half-storage matrix based on the two-dimensional row and column indices provided.

3.3.6.2.9.4.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Value	Elements	In	New value to be placed in Matrix
Row	Row_Indices	In	Row where New_Value is to be placed
Col	Col_Indices	In	Column where New_Value is to be placed
Matrix	Matrices	Out	Half-storage matrix to be updated

3.3.6.2.9.4.10.6.4 LOCAL DATA

None.

3.3.6.2.9.4.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Change_Element (New_Value : in     Elements;
                          Row       : in     Row_Indices;
                          Col       : in     Col_Indices;
                          Matrix    :    out Matrices) is

begin
```

```
--    --determine which half of the matrix is being referenced
      if Row_Indices'POS(Row) - Row_Indices'POS(Row_Indices'FIRST) >=
         Col_Indices'POS(Col) - Col_Indices'POS(Col_Indices'FIRST) then

--       --looking at bottom half of array
         Matrix(Row_Marker(Row) + Col_Offset(Col)) := New_Value;

      else

--       --looking at top half; need to switch to bottom half
         Matrix(Row_Marker(Swap_Row(Col)) +
                Col_Offset(Swap_Col(Row))) := New_Value;

      end if;

   end Change_Element;
```

3.3.6.2.9.4.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Swap_Row	function	Takes a column as input, and returns the corresponding row entry
Swap_Col	function	Takes a row as input, and returns the corresponding column entry

3.3.6.2.9.4.10.6.8 LIMITATIONS

None.

3.3.6.2.9.4.10.7 RETRIEVE_ELEMENT UNIT DESIGN (CATALOG #P383-0)

This function retrieves an element from a half-storage matrix using two-dimensional row and column indices as input.

3.3.6.2.9.4.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Half-storage matrix containing desired element
Row	Row_Indices	In	Row in which element is contained

3.3.6.2.9.4.10.7.4 LOCAL DATA

None.

3.3.6.2.9.4.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
   function Retrieve_Element (Matrix : Matrices;
                              Row    : Row_Indices;
                              Col    : Col_Indices) return Elements is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Elements;

-- -------------------------------
-- --begin function Retrieve_Element
-- -------------------------------

   begin

--    --determine which half of the array is being referenced
      if Row_Indices'POS(Row) - Row_Indices'POS(Row_Indices'FIRST) >=
         Col_Indices'POS(Col) - Col_Indices'POS(Col_Indices'FIRST) then

--       --already looking at the bottom half of the array
         Answer := Matrix(Row_Marker(Row) + Col_Offset(Col));

      else

--       --looking at the top half; need to switch to bottom half
         Answer := Matrix(Row_Marker(Swap_Row(Col)) +
                          Col_Offset(Swap_Col(Row)));

      end if;

      return Answer;
```

```
   end Retrieve_Element;
```

3.3.6.2.9.4.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Swap_Row	function	Takes a column as input, and returns the corresponding row entry
Swap_Col	function	Takes a row as input, and returns the corresponding column entry

3.3.6.2.9.4.10.7.8 LIMITATIONS

None.

3.3.6.2.9.4.10.8 ROW_SLICE UNIT DESIGN (CATALOG #P384-0)

This function returns an array which contains all the elements in a requested row.

3.3.6.2.9.4.10.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.8.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Half-storage matrix containing values to be retrieved
Row	Row_Indices	In	Indicates which row of values is desired

3.3.6.2.9.4.10.8.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Row_Slices	N/A	Requested row of values

3.3.6.2.9.4.10.8.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.8.6 PROCESSING

The following describes the processing performed by this part:

```
   function Row_Slice (Matrix : Matrices;
                       Row    : Row_Indices) return Row_Slices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Row_Slices;

-- -------------------------
-- --begin function Row_Slice
-- -------------------------

   begin

--    --retrieve row elements in bottom half of array
      Bottom_Loop:
         for Col in Col_Indices'FIRST .. Swap_Col(Row) loop
            Answer(Col) := Matrix(Row_Marker(Row) + Col_Offset(Col));
         end loop Bottom_Loop;

--    --retrieve row elements in top half of array, if there are any
      if Row /= Row_Indices'LAST then
         Top_Loop:
            for Col in Col_Indices'SUCC(Swap_Col(Row)) .. Col_Indices'LAST loop
               Answer(Col) := Matrix(Row_Marker(Swap_Row(Col)) +
                                     Col_Offset(Swap_Col(Row)));
            end loop Top_Loop;
      end if;

      return Answer

   end Row_Slice;
```

3.3.6.2.9.4.10.8.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored
Row_Slices	array	Data type defining a row slice of a matrix

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following data objects are required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Swap_Row	function	Takes a column as input, and returns the corresponding row entry
Swap_Col	function	Takes a row as input, and returns the corresponding column entry

3.3.6.2.9.4.10.8.8 LIMITATIONS

None.

3.3.6.2.9.4.10.9 COLUMN_SLICE UNIT DESIGN (CATALOG #P385-0)

This function retrieves all the values contained in a single column of a symmetric matrix.

3.3.6.2.9.4.10.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.9.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.9.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Half-storage matrix from which a column of values is to be retrieved
Col	Col_Indices	In	Indicates which column of values is desired

3.3.6.2.9.4.10.9.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Col_Slices	N/A	Column's worth of values

3.3.6.2.9.4.10.9.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.9.6 PROCESSING

The following describes the processing performed by this part:

```
   function Column_Slice (Matrix : Matrices;
                          Col    : Col_Indices) return Col_Slices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Col_Slices;

-- ----------------------------
-- --begin function Column_Slice
-- ----------------------------

   begin

--    --retrieve column elements contained in bottom half of array
      Bottom_Loop:
         for Row in Swap_Row(Col) .. Row_Indices'LAST loop
            Answer(Row) := Matrix(Row_Marker(Row) + Col_Offset(Col));
         end loop Bottom_Loop;

--    --retrieve column elements contained in top half of array, if any
      if Col /= Col_Indices'FIRST then
         Top_Loop:
            for Row in Row_Indices'FIRST .. Row_Indices'PRED(Swap_Row(Col)) loop
               Answer(Row) := Matrix(Row_Marker(Swap_Row(Col)) +
                                     Col_Offset(Swap_Col(Row)));
            end loop Top_Loop;
      end if;

      return Answer;

   end Column_Slice;
```

3.3.6.2.9.4.10.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored
Col_Slices	array	Data type defining a column slice of a matrix

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Swap_Row	function	Takes a column as input, and returns the corresponding row entry
Swap_Col	function	Takes a row as input, and returns the corresponding column entry

3.3.6.2.9.4.10.9.8 LIMITATIONS

None.

3.3.6.2.9.4.10.10 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P386-0)

This function adds an input matrix to an identity matrix, returning the result. The addition is performed by adding 1.0 to each diagonal element of the input matrix.

3.3.6.2.9.4.10.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.10.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.10.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Matrix to be added to an identity matrix

3.3.6.2.9.4.10.10.4 LOCAL DATA

Data objects:

The following table summarizes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Result of adding input matrix to identity matrix

3.3.6.2.9.4.10.10.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.10.6 PROCESSING

The following describes the processing performed by this part:

```
   function Add_to_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Matrices;

-- --------------------------------
-- --begin function Add_To_Identity
-- --------------------------------

   begin

--    --do straight assignment of all elements and then add in the
--    --identity matrix

      Answer := Input;

--    --all diagonal elements, except for the last one, are located one
--    --entry before the starting location of the next row
      Add_Identity_Loop:
         for Index in Row_Indices'SUCC(Row_Indices'FIRST) ..
                      Row_Indices'LAST loop
            Answer(Row_Marker(Index) - 1) := Answer(Row_Marker(Index)-1) + 1.0;
         end loop Add_Identity_Loop;

--    --handle last diagonal element
      Answer(Entry_Count) := Answer(Entry_Count) + 1.0;

      return Answer;

   end Add_to_Identity;
```

3.3.6.2.9.4.10.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Value	Description
Entry_Count	Positive	---	Number of stored values from the half-storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

3.3.6.2.9.4.10.10.8 LIMITATIONS

None.

3.3.6.2.9.4.10.11 SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P387-0)

This function subtracts an input matrix from an identity matrix. It does this by first subtracting the input matrix from a zero matrix and then adding it to an identity matrix.

3.3.6.2.9.4.10.11.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.11.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.11.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Matrix to be subtracted from an identity matrix

3.3.6.2.9.4.10.11.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Result of subtracting input matrix from an identity matrix

3.3.6.2.9.4.10.11.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.11.6 PROCESSING

The following describes the processing performed by this part:

```
   function Subtract_from_Identity (Input : Matrices) return Matrices is

--    --------------------
```

```
--    --declaration section
--    ---------------------

      Answer : Matrices;

-- ---------------------------------------
-- --begin function Subtract_from_Identity
-- ---------------------------------------

   begin

--    --subtract Input from a zero matrix and then add it to an identity matrix

      Subtract_Loop:
         for Index in 1..Entry_Count loop
            Answer(Index) := - Input(Index);
         end loop Subtract_Loop;

--    --all diagonal elements, except for the last one, are located one
--    --entry before the starting location of the next row
      Add_Identity_Loop:
         for Index in Row_Indices'SUCC(Row_Indices'FIRST) ..
                      Row_Indices'LAST loop
            Answer(Row_Marker(Index) - 1) := Answer(Row_Marker(Index)-1) + 1.0;
         end loop Add_Identity_Loop;

--    --handle last diagonal element
      Answer(Entry_Count) := Answer(Entry_Count) + 1.0;

      return Answer;

   end Subtract_from_Identity;
```

3.3.6.2.9.4.10.11.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

PARAMETERS:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Value	Description
Entry_Count	Positive	---	Number of stored values from the half-storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

3.3.6.2.9.4.10.11.8 LIMITATIONS

None.

3.3.6.2.9.4.10.12 "+" UNIT DESIGN (CATALOG #P388-0)

This function adds two half-storage matrices by adding the individual elements of the input matrices, returning the resultant matrix.

3.3.6.2.9.4.10.12.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.12.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.12.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	First matrix to be added
Right	Matrices	In	Second matrix to be added

3.3.6.2.9.4.10.12.4 LOCAL DATA

Data objects:

The following data objects are maintained by this part:

Name	Type	Description
Answer	Matrices	Result of adding the two input matrices

3.3.6.2.9.4.10.12.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.12.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    --------------------
--    --declaration section
```

```
--     ---------------------

       Answer : Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Process:
         for Index in     ntry_Count loop
            Answer(Index) := Left(Index) + Right(Index);
         end loop Process;

      return Answer;

   end "+";
```

3.3.6.2.9.4.10.12.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Value	Description
Entry_Count	Positive	---	Number of stored values from the half-storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Swap_Row	function	Takes a column as input, and returns the corresponding row entry
Swap_Col	function	Takes a row as input, and returns the corresponding column entry

3.3.6.2.9.4.10.12.8 LIMITATIONS

None.

3.3.6.2.9.4.10.13 "-" UNIT DESIGN (CATALOG #P389-0)

This function subtracts two half-storage matrices by subtracting the individual elements of the input matrices, returning the resultant matrix.

3.3.6.2.9.4.10.13.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.13.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.13.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Matrix to be subtracted from
Right	Matrices	In	Matrix to be subtracted from Left

3.3.6.2.9.4.10.13.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of subtracting Right input matrix from Left input matrix

3.3.6.2.9.4.10.13.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.13.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    --------------------
--    --declaration section
--    --------------------

      Answer : Matrices;

-- -------------------
```

```
-- --begin function "-"
-- --------------------

   begin

      Process:
         for Index in 1 .. Entry_Count loop
            Answer(Index) := Left(Index) - Right(Index);
         end loop Process;

      return Answer;

   end "-";
```

3.3.6.2.9.4.10.13.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Value	Description
Entry_Count	Positive	---	Number of stored values from the half-storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description
Col_Offset	Col_Index_Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_Arrays	Used to marker where in Matrices a particular row begins

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

Name	Type	Description
Swap_Row	function	Takes a column as input, and returns the corresponding row entry
Swap_Col	function	Takes a row as input, and returns the corresponding column entry

3.3.6.2.9.4.10.13.8 LIMITATIONS

None.

3.3.6.2.9.5 SYMMETRIC_FULL_STORAGE_MATRIX_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P390-0)

This package exports operations on a symmetric full storage matrix.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level:

Data types:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

3.3.6.2.9.5.4 LOCAL DATA

Exceptions:

The following table describes the exceptions defined in this part's package specification:

Name	Description
Invalid_Index	Indicates an attempt was made to access an element beyond the dimensions of the array

Data types:

The following types are previously defined in this part's package specification:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Symmetric_Full_Storage_Matrix_Operations_Unconstrained is

end Symmetric_Full_Storage_Matrix_Operations_Unconstrained;
```

3.3.6.2.9.5.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.5.8 LIMITATIONS

None.

3.3.6.2.9.5.9 LLCSC DESIGN

None.

3.3.6.2.9.5.10 UNIT DESIGN

3.3.6.2.9.5.10.1 CHANGE_ELEMENT UNIT DESIGN (CATALOG #P391-0)

This procedure changes the indicated element of a symmetric matrix, along with its symmetric counterpart.

3.3.6.2.9.5.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Value	Elements	In	New value to be placed in the matrix
Row	Row_Indices	In	Row in which the value belongs
Col	Col_Indices	In	Column in which the value belongs
Matrix	Matrices	In/Out	Matrix being updated

3.3.6.2.9.5.10.1.4 LOCAL DATA

None.

3.3.6.2.9.5.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Change_Element (New_Value : in     Elements;
                             Row       : in     Row_Indices;
                             Col       : in     Col_Indices;
                             Matrix    : in out Matrices) is

--    ----------------------
--    --declaration section-
--    ----------------------

      S_Col : Col_Indices;
      S_Row : Row_Indices;

-- ---------------------------------
-- --begin procedure Change_Element-
-- ---------------------------------

   begin

--    --make sure you have a square matrix
      if Matrix'LENGTH(1) /= Matrix'LENGTH(2) then

         raise Dimension_Error;

--    --make sure row and col are within bounds
      elsif NOT (Row in Matrix'RANGE(1)  and
                 Col in Matrix'RANGE(2)) then

         raise Invalid_Index;

      else

--       --everything is okay
```

```
         S_Col := Col_Indices'VAL(Row_Indices'POS(Row) -
                                  Row_Indices'POS(Matrix'FIRST(1)) +
                                  Col_Indices'POS(Matrix'FIRST(2)));

         S_Row := Row_Indices'VAL(Col_Indices'POS(Col) -
                                  Col_Indices'POS(Matrix'FIRST(2)) +
                                  Row_Indices'POS(Matrix'FIRST(1)));

         Matrix(Row, Col)     := New_Value;
         Matrix(S_Row, S_Col) := New_Value;

      end if;

   end Change_Element;
```

3.3.6.2.9.5.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package body of Symmetric_Full_Storage_Matrix_Operations_Unconstrained:

Name	Description
Invalid_Index	Indicates an attempt was made to access an element beyond the dimensions of the array

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine or package when input received has dimensions incompatible with the type of operation to be performed

3.3.6.2.9.5.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Index	Raised if an attempt is made to place an element outside the bounds of the input array
Dimension_ Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.5.10.2 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P392-0)

This procedure turns an input matrix into an identity matrix. An identity matrix is one where all elements equal 0.0, except those on the diagonal which equal 1.0. The input matrix must be a square matrix, but the ranges of the individual dimensions do not have to be the same.

3.3.6.2.9.5.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	Out	Matrix to be made into an identity matrix

3.3.6.2.9.5.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Col	Col_Indices	N/A	Index into second dimension of input matrix
Row	Row_Indices	N/A	Index into first dimension of inut matrix

3.3.6.2.9.5.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

-- ----------------------
-- --declaration section-
-- ----------------------

   Col : Col_Indices;
   Row : Row_Indices;

-- ---------------------------------
-- --begin procedure Set_to_Identity-
-- ---------------------------------

   begin

--     --make sure input matrix is a square matrix
       if Matrix'LENGTH(1) = Matrix'LENGTH(2) then


          Matrix := (others => (others => 0.0));

          Row := Matrix'FIRST(1);
          Col := Matrix'FIRST(2);
```

```
         Row_Loop:
            loop

--             --set diagonal element equal to
               Matrix(Row, Col) := 1.0;

               exit Row_Loop when Row = Matrix'LAST(1);
               Row := Row_Indices'SUCC(Row);
               Col := Col_Indices'SUCC(Col);

            end loop Row_Loop;

      else

--       --do not have a square matrix
         raise Dimension_Error;

      end if;

   end Set_To_Identity_Matrix;
```

3.3.6.2.9.5.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.5.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if input matrix is a square matrix

3.3.6.2.9.5.10.3 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P393-0)

This procedure zeros out all elements of an input matrix.

3.3.6.2.9.5.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	Out	Matrix to be zeroed out

3.3.6.2.9.5.10.3.4 LOCAL DATA

None.

3.3.6.2.9.5.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

begin

   Matrix := (others => (others => 0.0));

end Set_To_Zero_Matrix;
```

3.3.6.2.9.5.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.5.10.3.8 LIMITATIONS

None.

3.3.6.2.9.5.10.4 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P394-0)

This function adds an input matrix to an identity matrix, returning the resultant matrix. The addition is performed by adding 1.0 to each diagonal element of the input matrix.

3.3.6.2.9.5.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Matrix to be added to an identity matrix

3.3.6.2.9.5.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding identity matrix to input matrix
Col	Col_Indices	N/A	Index into second dimension of matrices
Row	Row_Indices	N/A	Index into first dimension of matrices

3.3.6.2.9.5.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function Add_to_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer     : Matrices(Input'RANGE(1), Input'RANGE(2));
      Col        : Col_Indices;
      Row        : Row_Indices;

-- --------------------------------
-- --begin function Add_to_Identity
-- --------------------------------

   begin

--    --make sure input matrix is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then

         Answer := Input;

         Row := Input'FIRST(1);
         Col := Input'FIRST(2);
         Access_Diagonal_Elements:
            loop

               Answer(Row,Col) := Answer(Row,Col) + 1.0;

               exit Access_Diagonal_Elements when Row = Input'LAST(1);
               Row := Row_Indices'SUCC(Row);
               Col := Col_Indices'SUCC(Col);

            end loop Access_Diagonal_Elements;

      else

--       --do not have a square matrix
         raise Dimension_Error;

      end if;

      return Answer;

   end Add_to_Identity;
```

3.3.6.2.9.5.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.5.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if input matrix is not a square matrix

3.3.6.2.9.5.10.5 SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P395-0)

This function subtracts an input matrix from an identity matrix, returning the resultant matrix.

3.3.6.2.9.5.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Matrix to be subtracted from an identity matrix

3.3.6.2.9.5.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of subtracting input matrix from an identity matrix
Col	Col_Indices	N/A	Used to index second dimension of matrices
Col_Count	POSITIVE	N/A	Used to count the number of columns accessed; needed to determine when the diagonal element has been reached
Row	Row_Indices	N/A	Used to index second dimension of matrices
Row_Count	POSITIVE	N/A	Used to count the number of rows accessed; when Col_Count equals this the diagonal element has been reached
S_Col	Col_Indices	N/A	Used to mark the column containing the diagonal element for the current row
S_Row	Row_Indices	N/A	Used in conjunction with S_Col to locate the symmetric counterpart to the element being referenced in the bottom half of the matrix

3.3.6.2.9.5.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
   function Subtract_from_Identity (Input : Matrices) return Matrices is

--    --------------------
--    --declaration section
--    --------------------

      Answer    : Matrices(Input'RANGE(1), Input'RANGE(2));
      Col       : Col_Indices;
      Col_Count : POSITIVE;
      Row       : Row_Indices;
      Row_Count : POSITIVE;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- ---------------------------------------
-- --begin function Subtract_from_Identity
-- ---------------------------------------

   begin

--    --make sure input matrix is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then

--       --will subtract input matrix from an identity matrix by first
--       --subtracting all elements from 0.0 and then adding 1.0 to the
--       --diagonal elements;
--       --when doing the subtraction, will only calculate the remainder
--       --for the elements in the bottom half of the matrix and will simply
--       --do assignments for the symmetric elements in the top half of the
--       --matrix

         Row_Count := 1;

--       --S_Col will go across the columns as Row goes down the rows;
--       --will mark column containing the diagonal element for this row
         Row   := Input'FIRST(1);
         S_Col := Input'FIRST(2);
         Do_Every_Row:
           loop

              Col_Count := 1;

--            --S_Row will go down the rows as Col goes across the columns;
--            --when paired with S_Col will mark the symmetric counterpart
--            --to the element being referenced in the bottom half of the
--            --matrix
              Col       := Input'FIRST(2);
              S_Row     := Input'FIRST(1);
```

```
               Subtract_Elements_From_Zero:
                  loop

--                   --perform subtraction on element in bottom half of matrix
                     Answer(Row,Col) := - Input(Row,Col);

--                   --exit loop after diagonal element has been reached
                     exit Subtract_Elements_From_Zero when Col_Count =
                                                           Row_Count;

--                   --assign values to symmetric elements in top half of matrix
--                   --(done after check for diagonal, since diagonal elements
--                   -- don't have a symmetric counterpart)
                     Answer(S_Row,S_Col) := Answer(Row,Col);

--                   --increment variables
                     Col_Count := Col_Count + 1;
                     Col       := Col_Indices'SUCC(Col);
                     S_Row     := Row_Indices'SUCC(S_Row);

                  end loop Subtract_Elements_From_Zero;

--             --add one to the diagonal element
               Answer(Row, Col) := Answer(Row, S_Col) + 1.0;

               exit Do_Every_Row when Row_Count = Input'LENGTH(1);
               Row_Count := Row_Count + 1;
               Row       := Row_Indices'SUCC(Row);
               S_Col     := Col_Indices'SUCC(S_Col);

            end loop Do_Every_Row;

      else

         raise Dimension_Error;

      end if;

      return Answer;

   end Subtract_from_Identity;
```

3.3.6.2.9.5.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.5.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if input matrix is not a square matrix

3.3.6.2.9.5.10.6 "+" UNIT DESIGN (CATALOG #P396-0)

This function adds two symmetric matrices by adding the individual elements of the input matrices, taking advantage of their symmetricity.

3.3.6.2.9.5.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	First matrix to be added
Right	Matrices	In	Second matrix to be added

3.3.6.2.9.5.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding input matrices
Row_Count, Col_Count	Positive	N/A	Used to keep track of the number of rows and columns which have been handled
L_Col	Col_Indices	N/A	Column index into left matrix
L_Row	Row_Indices	N/A	Row index into left matrix
R_Col	Col_Indices	N/A	Column index into right matrix
R_Row	Row_Indices	N/A	Row index into right matrix
S_Col	Col_Indices	N/A	New column index after row and column have been swapped, i.e. (i,j) => (j,i)
S_Row	Row_Indices	N/A	New row index after row and column have been swapped

3.3.6.2.9.5.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    --------------------
```

```
--    --declaration section
--    ---------------------

      Answer    : Matrices(Left'RANGE(1), Left'RANGE(2));
      Col_Count : POSITIVE;
      Row_Count : POSITIVE;
      L_Col     : Col_Indices;
      L_Row     : Row_Indices;
      R_Col     : Col_Indices;
      R_Row     : Row_Indices;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --make sure both input matrices are square matrices of the same size
      if Left'LENGTH(1)  = Left'LENGTH(2) and
         Left'LENGTH(1)  = Right'LENGTH(1) and
         Right'LENGTH(1) = Right'LENGTH(2) then

--       --addition calculations will only be carried out on the bottom half
--       --of the input matrices followed by assignments to the symmetric
--       --elements in the top half of the matrix

         Row_Count := 1;

--       --as L_Row goes down the rows, S_Col will go across the columns
         L_Row     := Left'FIRST(1);
         S_Col     := Left'FIRST(2);

         R_Row     := Right'FIRST(1);
         Do_All_Rows:
            loop

               Col_Count := 1;

--             --as L_Col goes across the columns, S_Row will go down the rows
               L_Col := Left'FIRST(2);
               S_Row := Left'FIRST(1);

               R_Col := Right'FIRST(2);
               Add_Bottom_Half_Elements:
                  loop

                     Answer(L_Row,L_Col) := Left(L_Row, L_Col) +
                                            Right(R_Row, R_Col);

--                   --exit when diagonal element has been reached
                     exit Add_Bottom_Half_Elements when Col_Count = Row_Count;

--                   --assign value to symmetric element in top half of matrix
--                   --(do this after exit since diagonal elements don't have
--                   -- a corresponding symmetric element)
```

```
                    Answer(S_Row,S_Col) := Answer(L_Row,L_Col);

--                  --increment values
                    Col_Count := Col_Count + 1;
                    L_Col     := Col_Indices'SUCC(L_Col);
                    S_Row     := Row_Indices'SUCC(S_Row);
                    R_Col     := Col_Indices'SUCC(R_Col);

                end loop Add_Bottom_Half_Elements;

            exit Do_All_Rows when Row_Count = Left'LENGTH(1);
            Row_Count := Row_Count + 1;
            L_Row     := Row_Indices'SUCC(L_Row);
            S_Col     := Col_Indices'SUCC(S_Col);
            R_Row     := Row_Indices'SUCC(R_Row);

        end loop Do_All_Rows;

    else

        raise Dimension_Error;

    end if;

    return Answer;

end "+";
```

3.3.6.2.9.5.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.5.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if input matrices are not both m x m matrices

3.3.6.2.9.5.10.7 "-" UNIT DESIGN (CATALOG #P397-0)

This function subtracts two symmetric input matrices by subtracting the individual elements of the input matrices, taking advantage of their symmetricity.

3.3.6.2.9.5.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Matrix to be subtracted from
Right	Matrices	In	Matrix to be used as the subtrahend

3.3.6.2.9.5.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding input matrices
Row_Count, Col_Count	Positive	N/A	Used to keep track of the number of rows and columns which have been handled
L_Col	Col_Indices	N/A	Column index into left matrix
L_Row	Row_Indices	N/A	Row index into left matrix
R_Col	Col_Indices	N/A	Column index into right matrix
R_Row	Row_Indices	N/A	Row index into right matrix
S_Col	Col_Indices	N/A	New column index after row and column have been swapped, i.e. (i,j) => (j,i)
S_Row	Row_Indices	N/A	New row index after row and column have been swapped

3.3.6.2.9.5.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer    : Matrices(Left'RANGE(1), Left'RANGE(2));
      Col_Count : POSITIVE;
      Row_Count : POSITIVE;
      L_Col     : Col_Indices;
      L_Row     : Row_Indices;
      R_Col     : Col_Indices;
      R_Row     : Row_Indices;
      S_Col     : Col_Indices;
```

```
      S_Row     : Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --make sure both input matrices are square matrices of the same size
      if Left'LENGTH(1)  = Left'LENGTH(2) and
         Left'LENGTH(1)  = Right'LENGTH(1) and
         Right'LENGTH(1) = Right'LENGTH(2) then

--       --addition calculations will only be carried out on the bottom half
--       --of the input matrices followed by assignments to the symmetric
--       --elements in the top half of the matrix

         Row_Count := 1;

--       --as L_Row goes down the rows, S_Col will go across the columns
         L_Row     := Left'FIRST(1);
         S_Col     := Left'FIRST(2);

         R_Row     := Right'FIRST(1);
         Do_All_Rows:
            loop

               Col_Count := 1;

--             --as L_Col goes across the columns, S_Row will go down the rows
               L_Col := Left'FIRST(2);
               S_Row := Left'FIRST(1);

               R_Col := Right'FIRST(2);
               Add_Bottom_Half_Elements:
                  loop

                     Answer(L_Row,L_Col) := Left(L_Row, L_Col) -
                                            Right(R_Row, R_Col);

--                   --exit when diagonal element has been reached
                     exit Add_Bottom_Half_Elements when Col_Count = Row_Count;

--                   --assign value to symmetric element in top half of matrix
--                   --(do this after exit since diagonal elements don't have
--                   -- a corresponding symmetric element)
                     Answer(S_Row,S_Col) := Answer(L_Row,L_Col);

--                   --increment values
                     Col_Count := Col_Count + 1;
                     L_Col     := Col_Indices'SUCC(L_Col);
                     S_Row     := Row_Indices'SUCC(S_Row);
                     R_Col     := Col_Indices'SUCC(R_Col);

                  end loop Add_Bottom_Half_Elements;

               exit Do_All_Rows when Row_Count = Left'LENGTH(1);
```

```
                Row_Count := Row_Count + 1;
                L_Row     := Row_Indices'SUCC(L_Row);
                S_Col     := Col_Indices'SUCC(S_Col);
                R_Row     := Row_Indices'SUCC(R_Row);

            end loop Do_All_Rows;

      else

         raise Dimension_Error;

      end if;

      return Answer;

   end "-";
```

3.3.6.2.9.5.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.5.10.7.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if input matrices are not both m x m matrices

3.3.6.2.9.6 DIAGONAL_MATRIX_OPERATIONS PACKAGE DESIGN (CATALOG #P408-0)

This package contains a set of functions designed to operate on a diagonal matrix.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.2 LOCAL ENTITIES DESIGN

Subprograms:

This package body contains a sequence of statements which are executed when it is elaborated. This code first checks to ensure a square matrix has been instantiated. If not, a Dimension_Error exception is raised. If a square matrix has been instantiated, then the Row_Marker and Col_Marker arrays are initialized.

3.3.6.2.9.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters are defined in this part's package specification:

Data types:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays
Col_Slices	array	One-dimensional array of column Elements
Row_Slices	array	One-dimensional array of row Elements

3.3.6.2.9.6.4 LOCAL DATA

Exceptions:

The following table describes the exceptions defined by this part's package specification:

Name	Description
Invalid_Index	Indicates an attempt was made to access an element not on the diagonal

Data types:

The following table describes the data types defined in this part's package specification:

Name	Range	Description
Diagonal_Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

The following table describes the data types defined by this part:

Name	Range	Description
Row_Markers	N/A	Array of pointers into Diagonal_Matrices
Col_Markers	N/A	Array of pointers into Diagonal_Matrices

Data objects:

The following table describes the data objects defined in this part's package specification:

Name	Type	Description
Entry_Count	Positive	Number of diagonal elements in the array

The following table describes the data objects maintained by this part:

Name	Type	Description
Row_Marker	Row_Markers	Array of pointers into Diagonal_ Matrices
Col_Marker	Col_Markers	Array of pointers into Diagonal_ Matrices
Row_Minus_Col_Indices_ Pos_First	INTEGER	Preinitialized value of: Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)
Local_Identity_Matrix	Diagonal_ Matrices	Pre-initialized identity matrix
Local_Zero_Matrix	Diagonal_ Matrices	Pre-initialized zero matrix

The following table describes the data objects contained in the declare block located at the end of this package body:

Name	Type	Value	Description
Col_Count	Positive	N/A	Counts the number of columns
Row_Count	Positive	N/A	Counts the nubmer of rows

3.3.6.2.9.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Diagonal_Matrix_Operations is

-- --------------------
-- --local declarations
-- --------------------

   type Col_Markers is array(Col_Indices) of POSITIVE;
   type Row_Markers is array(Row_Indices) of POSITIVE;
```

```
   Col_Marker : Col_Markers;
   Row_Marker : Row_Markers;

   Row_Minus_Col_Indices_Pos_First : constant INTEGER
                                     := Row_Indices'POS(Row_Indices'FIRST) -
                                        Col_Indices'POS(Col_Indices'FIRST);

   Local_Identity_Matrix : constant Diagonal_Matrices := (others => 1.0);
   Local_Zero_Matrix     : constant Diagonal_Matrices := (others => 0.0);


--------------------------------
--begin processing for Diagonal_
--Matrix_Operations package
--------------------------------
begin

   Init_Block:
      declare

         Col_Count : POSITIVE;
         Row_Count : POSITIVE;

      begin

--       --make sure lengths of indices are the same
         if Row_Slices'LENGTH = Col_Slices'LENGTH then

--          --initialize row and column marker arrays

            Row_Count := 1;
            Row_Init:
               for Row in Row_Indices loop
                  Row_Marker(Row) := Row_Count;
                  Row_Count       := Row_Count + 1;
               end loop Row_Init;

            Col_Count := 1;
            Col_Init:
               for Col in Col_Indices loop
                  Col_Marker(Col) := Col_Count;
                  Col_Count       := Col_Count + 1;
               end loop Col_Init;

         else

            raise Dimension_Error;

         end if;

      end Init_Block;

end Diagonal_Matrix_Operations;
```

3.3.6.2.9.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of Row_Indices and Col_Indices are not the same

3.3.6.2.9.6.9 LLCSC DESIGN

None.

3.3.6.2.9.6.10 UNIT DESIGN

3.3.6.2.9.6.10.1 IDENTITY_MATRIX UNIT DESIGN (CATALOG #P409-0)

This function returns a diagonal matrix which has been set to an identity matrix. An identity matrix is one where all the elements equal 0.0, except for the diagonal elements which equal 1.0.

3.3.6.2.9.6.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.1.3 INPUT/OUTPUT

None.

3.3.6.2.9.6.10.1.4 LOCAL DATA

None.

3.3.6.2.9.6.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Identity_Matrix return Diagonal_Matrices is

begin

   return Local_Identity_Matrix;

end Identity_Matrix;
```

3.3.6.2.9.6.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_ Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Type	Description
Local_Identity_Matrix	Diagonal_ Matrices	Pre-initialized identity matrix

3.3.6.2.9.6.10.1.8 LIMITATIONS

None.

3.3.6.2.9.6.10.2 ZERO_MATRIX UNIT DESIGN (CATALOG #P410-0)

This function returns a diagonal matrix which has been set to a zero matrix.

3.3.6.2.9.6.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.2.3 INPUT/OUTPUT

None.

3.3.6.2.9.6.10.2.4 LOCAL DATA

None.

3.3.6.2.9.6.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function Zero_Matrix return Diagonal_Matrices is

begin

   return Local_Zero_Matrix;

end Zero_Matrix;
```

3.3.6.2.9.6.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_ Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Type	Description
Local_Zero_Matrix	Diagonal_ Matrices	Pre-initialized zero matrix

3.3.6.2.9.6.10.2.8 LIMITATIONS

None.

3.3.6.2.9.6.10.3 CHANGE_ELEMENT UNIT DESIGN (CATALOG #P411-0)

This procedure changes a single element of a diagonal matrix.

3.3.6.2.9.6.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Value	Elements	In	New value to be placed in matrix
Row	Row_Indices	In	Row where value is to be placed
Col	Col_Indices	In	Column where value is to be placed
Matrix	Diagonal_ Matrices	Out	Diagonal matrix to be updated

3.3.6.2.9.6.10.3.4 LOCAL DATA

None.

3.3.6.2.9.6.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Change_Element (New_Value : in     Elements;
                             Row       : in     Row_Indices;
                             Col       : in     Col_Indices;
                             Matrix    :    out Diagonal_Matrices) is

   begin

--     --make sure element referenced is on the diagonal
       if Row_Marker(Row) = Col_Marker(Col) then

          Matrix(Row_Marker(Row)) := New_Value;

       else

          raise Invalid_Index;

       end if;

   end Change_Element;
```

3.3.6.2.9.6.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

The following table summarizes the types required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Range	Description
Row_Markers	N/A	Array of pointers into Diagonal_Matrices
Col_Markers	N/A	Array of pointers into Diagonal_Matrices

The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Type	Description
Row_Marker	Row_Markers	Array of pointers into Diagonal_Matrices
Col_Marker	Col_Markers	Array of pointers into Diagonal_Matrices

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Description
Invalid_Index	Indicates an attempt was made to access an element not on the diagonal

3.3.6.2.9.6.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Index	Raised if in Row and Col indices do not fall on the diagonal

3.3.6.2.9.6.10.4 RETRIEVE_ELEMENT UNIT DESIGN (CATALOG #P412-0)

This function returns an element contained in a diagonal matrix.

3.3.6.2.9.6.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Diagonal_Matrices	In	Diagonal matrix containing element desired
Row	Row_Indices	In	Row containing element desired
Col	Col_Indices	In	Column containing element desired

3.3.6.2.9.6.10.4.4 LOCAL DATA

None.

3.3.6.2.9.6.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function Retrieve_Element (Matrix  : Diagonal_Matrices;
                              Row     : Row_Indices;
                              Col     : Col_Indices) return Elements is

   begin

--    --make sure (row,col) falls on the diagonal
      if Row_Marker(Row) /= Col_Marker(Col) then
         raise Invalid_Index;
      end if;

      return Matrix(Row_Marker(Row));
```

```
   end Retrieve_Element;
```

3.3.6.2.9.6.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_ Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

The following table summarizes the types required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Range	Description
Row_Markers	N/A	Array of pointers into Diagonal_ Matrices
Col_Markers	N/A	Array of pointers into Diagonal_ Matrices

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Type	Description
Row_Marker	Row_Markers	Array of pointers into Diagonal_ Matrices
Col_Marker	Col_Markers	Array of pointers into Diagonal_ Matrices

3.3.6.2.9.6.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Index	Raised if Row and Col indices do not fall on the diagonal

3.3.6.2.9.6.10.5 ROW_SLICE UNIT DESIGN (CATALOG #P413-0)

This function returns a row of values from a diagonal matrix.

3.3.6.2.9.6.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Diagonal_ Matrices	In	Diagonal matrix containing row desired
Row	Row_Indices	In	Row desired

3.3.6.2.9.6.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Col_Spot	Col_Indices	N/A	Index into Answer
Answer	Row_Slices	N/A	Row of values

3.3.6.2.9.6.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
   function Row_Slice (Matrix : Diagonal_Matrices;
                       Row    : Row_Indices) return Row_Slices is

--    ---------------------
--    --declaration section
--    ---------------------

      Col_Spot : Col_Indices;
      Answer   : Row_Slices;

-- -------------------------
-- --begin function Row_Slice
-- -------------------------

   begin

--    --zero out slice
      Answer := (others => 0.0);

--    --insert diagonal element
      Col_Spot := Col_Indices'VAL(Row_Indices'POS(Row) -
                                  Row_Minus_Col_Indices_Pos_First);
      Answer(Col_Spot) := Matrix(Row_Marker(Row));

      return Answer;

   end Row_Slice;
```

3.3.6.2.9.6.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays
Row_Slices	array	One-dimensional array of row Elements

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

The following table summarizes the types required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Range	Operators	Description
Row_Markers	N/A	N/A	Array of pointers into Diagonal_Matrices

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Type	Description
Col_Marker	Col_Markers	Array of pointers into Diagonal_Matrices
Row_Minus_Col_Indices_Pos_First	INTEGER	Preinitialized value of: Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)

3.3.6.2.9.6.10.5.8 LIMITATIONS

None.

3.3.6.2.9.6.10.6 COLUMN_SLICE UNIT DESIGN (CATALOG #P414-0)

This function returns a column's worth of values from a diagonal matrix.

3.3.6.2.9.6.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Diagonal_Matrices	In	Diagonal matrix containing desired column of values
Col	Col_Indices	In	Indicates which column is desired

3.3.6.2.9.6.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Col_Slices	N/A	Column of values from input matrix
Row_Spot	Row_Indices	N/A	Index into Answer

3.3.6.2.9.6.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
   function Column_Slice (Matrix : Diagonal_Matrices;
                          Col    : Col_Indices) return Col_Slices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer   : Col_Slices;
      Row_Spot : Row_Indices;

-- ----------------------------
-- --begin function Column_Slice
-- ----------------------------

   begin

--    --zero out answer and then insert diagonal value
      Answer := (others => 0.0);

--    --insert diagonal value
      Row_Spot := Row_Indices'VAL(Col_Indices'POS(Col) +
                                  Row_Minus_Col_Indices_Pos_First);
      Answer(Row_Spot) := Matrix(Col_Marker(Col));

      return Answer;

   end Column_Slice;
```

3.3.6.2.9.6.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays
Col_Slices	array	One-dimensional array of column Elements

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_ Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

The following table summarizes the types required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Range	Description
Col_Markers	N/A	Array of pointers into Diagonal_ Matrices

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Type	Description
Col_Marker	Col_Markers	Array of pointers into Diagonal_ Matrices
Row_Minus_Col_Indices_ Pos_First	INTEGER	Preinitialized value of: Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)

3.3.6.2.9.6.10.6.8 LIMITATIONS

None.

3.3.6.2.9.6.10.7 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P415-0)

This function adds the input matrix to an identity matrix, returning the resultant diagonal matrix. The calculations are performed by adding 1.0 to each diagonal element.

3.3.6.2.9.6.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Diagonal_Matrices	In	Diagonal matrix to be added to an identity matrix.

3.3.6.2.9.6.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Diagonal_Matrices	N/A	Result of performing addition

3.3.6.2.9.6.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
   function Add_to_Identity (Input : Diagonal_Matrices)
                             return Diagonal_Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Diagonal_Matrices;

-- --------------------------------
-- --begin function Add_to_Identity
-- --------------------------------

   begin

      Process:
```

```
         for Index in 1..Entry_Count loop
            Answer(Index) := Input(Index) + 1.0;
         end loop Process;

      return Answer;

   end Add_to_Identity;
```

3.3.6.2.9.6.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Type	Description
Entry_Count	Positive	Number of diagonal elements in the array

3.3.6.2.9.6.10.7.8 LIMITATIONS

None.

3.3.6.2.9.6.10.8 SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P416-0)

This function subtracts an input matrix from an identity matrix, returning the result. The calculations are performed by subtracting the diagonal elements from 1.0.

3.3.6.2.9.6.10.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.8.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Diagonal_Matrices	In	Diagonal matrix to be subtracted from an identity matrix

3.3.6.2.9.6.10.8.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Diagonal_Matrices	N/A	Result of performing the subtraction

3.3.6.2.9.6.10.8.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.8.6 PROCESSING

The following describes the processing performed by this part:

```
   function Subtract_from_Identity (Input : Diagonal_Matrices)
                                    return Diagonal_Matrices is

--    --------------------
--    --declaration section
--    --------------------

      Answer : Diagonal_Matrices;

-- ---------------------------------------
-- --begin function Subtract_From_Identity
-- ---------------------------------------

   begin

      Process:
         for Index in 1..Entry_Count loop
            Answer(Index) := 1.0 - Input(Index);
         end loop Process;

      return Answer;

   end Subtract_from_Identity;
```

3.3.6.2.9.6.10.8.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_ Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Type	Description
Entry_Count	Positive	Number of diagonal elements in the array

3.3.6.2.9.6.10.8.8 LIMITATIONS

None.

3.3.6.2.9.6.10.9 "+" (DIAGONAL_MATRICES + DIAGONAL_MATRICES => DIAGONAL_MATRICES) UNIT DESIGN (CATALOG #P417-0)

This function adds two input diagonal matrices, returning the resultant diagonal matrix.

3.3.6.2.9.6.10.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.9.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.9.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Diagonal_Matrices	In	First diagonal matrix to be added
Right	Diagonal_Matrices	In	Second diagonal matrix to be added

3.3.6.2.9.6.10.9.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Diagonal_Matrices	N/A	Result of performing the addition

3.3.6.2.9.6.10.9.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.9.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Diagonal_Matrices;
                 Right : Diagonal_Matrices) return Diagonal_Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Diagonal_Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Process:
         for Index in 1..Entry_Count loop
            Answer(Index) := Left(Index) + Right(Index);
         end loop Process;

      return Answer;

   end "+";
```

3.3.6.2.9.6.10.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Type	Description
Entry_Count	Positive	Number of diagonal elements in the array

3.3.6.2.9.6.10.9.8 LIMITATIONS

None.

3.3.6.2.9.6.10.10 "-" (DIAGONAL_MATRICES - DIAGONAL_MATRICES => DIAGONAL_MATRICES) UNIT DESIGN (CATALOG #P418-0)

This function subtracts two input diagonal matrices, returning the resultant matrix. The calculations are performed by subtracting the individual elements of the input matrices.

3.3.6.2.9.6.10.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.10.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.10.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Diagonal_Matrices	In	Diagonal matrix to be subtracted from
Right	Diagonal_Matrices	In	Diagonal matrix to be treated as the subtrahend

3.3.6.2.9.6.10.10.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Diagonal_Matrices	N/A	Result of performing the subtraction

3.3.6.2.9.6.10.10.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.10.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Diagonal_Matrices;
                 Right : Diagonal_Matrices) return Diagonal_Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Diagonal_Matrices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin
```

```
      Process:
         for Index in 1..Entry_Count loop
            Answer(Index) := Left(Index) - Right(Index);
         end loop Process;

      return Answer;

   end "-";
```

3.3.6.2.9.6.10.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_Range	1..Entry_Count	Used to dimension diagonal_matrices
Diagonal_Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Type	Description
Entry_Count	Positive	Number of diagonal elements in the array

3.3.6.2.9.6.10.10.8 LIMITATIONS

None.

3.3.6.2.9.7 VECTOR_SCALAR_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P419-0)

This package provides a set of functions to multiply and divide each element of a vector by a scalar.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.7.1 REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the units in this part:

Name	Requirements Allocation
"*"	R065
"/"	R066

3.3.6.2.9.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.7.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types previously in this part's package specification:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices1	discrete type	Used to dimension Vectors1
Indices2	discrete type	Used to dimension Vectors2
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars
"/"	function	Used to define the operation Elements2 := Elements1 / Scalars

3.3.6.2.9.7.4 LOCAL DATA

None.

3.3.6.2.9.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.7.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Scalar_Operations_Unconstrained is

end Vector_Scalar_Operations_Unconstrained;
```

3.3.6.2.9.7.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.7.8 LIMITATIONS

None.

3.3.6.2.9.7.9 LLCSC DESIGN

None.

3.3.6.2.9.7.10 UNIT DESIGN

3.3.6.2.9.7.10.1 "*" UNIT DESIGN (CATALOG #P420-0)

This function calculates a scaled vector by multiplying each element of an input vector by a scale factor.

3.3.6.2.9.7.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R065.

3.3.6.2.9.7.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.7.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Vectors2	In	Vector to be scaled
Multiplier	Scalars	In	Scale factor

3.3.6.2.9.7.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors1	Scaled vector
A_Index	Indices1	Index into answer array
V_Index	Indices2	Index into input vector

3.3.6.2.9.7.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.7.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Vector     : Vectors2;
                 Multiplier : Scalars) return Vectors1 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Vectors1(Indices1'FIRST ..
                         Indices1'VAL(Vector'LENGTH-1 +
                                      Indices1'POS(Indices1'FIRST) ));
      A_Index : Indices1;
      V_Index : Indices2;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

     A_Index := Indices1'FIRST;
     V_Index := Indices2'FIRST;
     Process:
        loop

           Answer(A_Index) := Vector(V_Index) * Multiplier;

           exit Process when V_Index = Vector'LAST;
           A_Index := Indices1'SUCC(A_Index);
           V_Index := Indices2'SUCC(V_Index);

        end loop Process;

     return Answer;

   end "*";
```

3.3.6.2.9.7.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Vector_Scalar_Operations_-Unconstrained:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices1	discrete type	Used to dimension Vectors1
Indices2	discrete type	Used to dimension Vectors2
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Vector_Scalar_Operations_Unconstrained package:

Name	Type	Description
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars

3.3.6.2.9.7.10.1.8 LIMITATIONS

None.

3.3.6.2.9.7.10.2 "/" UNIT DESIGN (CATALOG #P421-0)

This function calculates a scaled vector by dividing each element of an input vector by a scale factor.

3.3.6.2.9.7.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R066.

3.3.6.2.9.7.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.7.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Vectors1	In	Vector to be scaled
Divisor	Scalars	In	Scale factor

3.3.6.2.9.7.10.2.4 LOCAL DATA

Data objects:

The following describes the local data maintained by this part:

Name	Type	Description
Answer	Vectors2	Scaled vector
A_Index	Indices2	Index into answer vector
V_Index	Indices1	Index into input vector

3.3.6.2.9.7.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.7.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "/" (Vector  : Vectors1;
                 Divisor : Scalars) return Vectors2 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Vectors2(Indices2'FIRST ..
                         Indices2'VAL(Vector'LENGTH-1 +
                                      Indices2'POS(Indices2'FIRST) ));
      A_Index : Indices2;
      V_Index : Indices1;

-- ------------------------------------
-- --begin function Vector_Scalar_Divide
-- ------------------------------------

   begin
```

```
      A_Index := Indices2'FIRST;
      V_Index := Indices1'FIRST;
      Process:
         loop

            Answer(A_Index) := Vector(V_Index) / Divisor;

            exit Process when V_Index = Indices1'LAST;
            A_Index := Indices2'SUCC(A_Index);
            V_Index := Indices1'SUCC(V_Index);

         end loop Process;

      return Answer;

   end "/";
```

3.3.6.2.9.7.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Vector_Scalar_Operations_-Unconstrained:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices1	discrete type	Used to dimension Vectors1
Indices2	discrete type	Used to dimension Vectors2
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Vector_Scalar_Operations_Unconstrained package:

Name	Type	Description
"/"	function	Used to define the operation Elements2 := Elements1 / Scalars

3.3.6.2.9.7.10.2.8 LIMITATIONS

None.

3.3.6.2.9.8 MATRIX_SCALAR_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P425-0)

This package provides a set of functions which will scale a matrix by multiplying or dividing each element of the matrix by a scale factor.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.8.1 REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the parts in this LLCSC:

Name	Requirements Allocation
"*"	R073
"/"	R074

3.3.6.2.9.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.8.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described in this part's package specification:

Data types:

Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col_Indices1	discrete type	Used to dimension second dimension of Matrices1
Row_Indices1	discrete type	Used to dimension first dimension of Matrices1
Col_Indices2	discrete type	Used to dimension second dimension of Matrices2
Row_Indices2	discrete type	Used to dimension first dimension of Matrices2
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms:

Name	Type	Description
"*"	function	Function to define the operation Elements1 * Scalars := Elements2
"/"	function	Function to define the operation Elements2 / Scalars := Elements1

3.3.6.2.9.8.4 LOCAL DATA

None.

3.3.6.2.9.8.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.8.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Scalar_Operations_Unconstrained is

end Matrix_Scalar_Operations_Unconstrained;
```

3.3.6.2.9.8.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.8.8 LIMITATIONS

None.

3.3.6.2.9.8.9 LLCSC DESIGN

None.

3.3.6.2.9.8.10 UNIT DESIGN

3.3.6.2.9.8.10.1 "*" UNIT DESIGN (CATALOG #P426-0)

This function calculates a scaled matrix by multiplying each element of an input matrix by a scalar.

3.3.6.2.9.8.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R073.

3.3.6.2.9.8.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.8.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices1	In	Matrix to be scaled
Multiplier	Scalars	In	Scale factor

3.3.6.2.9.8.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices2	Scaled matrix
A_Col	Col_Indices2	Index into second dimension of answer matrix
A_Row	Row_Indices2	Index into first dimension of answer matrix
M_Col	Col_Indices1	Index into second dimension of input matrix
M_Row	Row_Indices1	Index into first dimension of input matrix

3.3.6.2.9.8.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.8.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Matrix     : Matrices1;
                 Multiplier : Scalars) return Matrices2 is

--   ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices2
                  (Row_Indices2'FIRST ..
                   Row_Indices2'VAL(Matrix'LENGTH(1)-1 +
                                    Row_Indices2'POS(Row_Indices2'FIRST) ),
                   Col_Indices2'FIRST ..
                   Col_Indices2'VAL(Matrix'LENGTH(2)-1 +
                                    Col_Indices2'POS(Col_Indices2'FIRST) ));
      A_Col  : Col_Indices2;
      A_Row  : Row_Indices2;
      M_Col  : Col_Indices1;
      M_Row  : Row_Indices1;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

      A_Row := Row_Indices2'FIRST;
      M_Row := Matrix'FIRST(1);
      Row_Loop:
         loop

            A_Col := Col_Indices2'FIRST;
            M_Col := Matrix'FIRST(2);
            Col_Loop:
               loop

                  Answer(A_Row, A_Col) := Matrix(M_Row, M_Col) * Multiplier;
```

```
                exit Col_Loop when M_Col = Matrix'LAST(2);
                A_Col := Col_Indices2'SUCC(A_Col);
                M_Col := Col_Indices1'SUCC(M_Col);

            end loop Col_Loop;

        exit Row_Loop when M_Row = Matrix'LAST(1);
        A_Row := Row_Indices2'SUCC(A_Row);
        M_Row := Row_Indices1'SUCC(M_Row);

    end loop Row_Loop;

  return Answer;

end "*";
```

3.3.6.2.9.8.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Matrix_Scalar_Operations:

Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col_Indices1	discrete type	Used to dimension second dimension of Matrices1
Row_Indices1	discrete type	Used to dimension first dimension of Matrices1
Col_Indices2	discrete type	Used to dimension second dimension of Matrices2
Row_Indices2	discrete type	Used to dimension first dimension of Matrices2
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Matrix_Scalar_Operations_Unconstrained package.

Name	Type	Description
"*"	function	Function to define the operation Elements1 * Scalars := Elements2

3.3.6.2.9.8.10.1.8 LIMITATIONS

None.

3.3.6.2.9.8.10.2 "/" UNIT DESIGN (CATALOG #P427-0)

This function calculates a scaled matrix by dividing each element of an input matrix by a scale factor.

3.3.6.2.9.8.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R074.

3.3.6.2.9.8.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.8.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices2	In	Matrix to be scaled
Divisor	Scalars	In	Scale factor

3.3.6.2.9.8.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices1	Scaled matrix
A_Col	Col_Indices1	Index into second dimension of answer matrix
A_Row	Row_Indices1	Index into first dimension of answer matrix
M_Col	Col_Indices2	Index into second dimension of input matrix
M_Row	Row_Indices2	Index into first dimension of input matrix

3.3.6.2.9.8.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.8.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "/" (Matrix  : Matrices2;
                 Divisor : Scalars) return Matrices1 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices1
                 (Row_Indices1'FIRST ..
                  Row_Indices1'VAL(Matrix'LENGTH(1)-1 +
                                   Row_Indices1'POS(Row_Indices1'FIRST) ),
                  Col_Indices1'FIRST ..
                  Col_Indices1'VAL(Matrix'LENGTH(2)-1 +
                                   Col_Indices1'POS(Col_Indices1'FIRST) ));
      A_Col  : Col_Indices1;
      A_Row  : Row_Indices1;
      M_Col  : Col_Indices2;
      M_Row  : Row_Indices2;

-- --------------------
-- --begin function "/"
-- --------------------

   begin

      A_Row := Row_Indices1'FIRST;
      M_Row := Matrix'FIRST(1);
      Row_Loop:
         loop

            A_Col := Col_Indices1'FIRST;
            M_Col := Matrix'FIRST(2);
            Col_Loop:
               loop

                  Answer(A_Row, A_Col) := Matrix(M_Row, M_Col) / Divisor;
```

```
                exit Col_Loop when M_Col = Matrix'LAST(2);
                A_Col := Col_Indices1'SUCC(A_Col);
                M_Col := Col_Indices2'SUCC(M_Col);

             end loop Col_Loop;

          exit Row_Loop when M_Row = Matrix'LAST(1);
          A_Row := Row_Indices1'SUCC(A_Row);
          M_Row := Row_Indices2'SUCC(M_Row);

       end loop Row_Loop;

    return Answer;

 end "/";
```

3.3.6.2.9.8.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Matrix_Scalar_Operations:

Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col_Indices1	discrete type	Used to dimension second dimension of Matrices1
Row_Indices1	discrete type	Used to dimension first dimension of Matrices1
Col_Indices2	discrete type	Used to dimension second dimension of Matrices2
Row_Indices2	discrete type	Used to dimension first dimension of Matrices2
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Matrix_Scalar_Operations_Unconstrained package.

Name	Type	Description
"*"	function	Function to define the operation Elements1 * Scalars := Elements2
"/"	function	Function to define the operation Elements2 / Scalars := Elements1

3.3.6.2.9.8.10.2.8 LIMITATIONS

None.

3.3.6.2.9.9 DIAGONAL_MATRIX_SCALAR_OPERATIONS PACKAGE DESIGN (CATALOG #P431-0)

This package provides the functions to allow the user to multiply or divide each element of a diagonal matrix by a scalar.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.9.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statements which are executed when it is elaborated. This code checks to ensure a square matrix has been instantiated. If not, a Dimension_Error exception is raised.

3.3.6.2.9.9.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in this part's package specification:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements1	floating point type	Type of elements in Diagonal_Matrices1
Elements2	floating point type	Type of elements in Diagonal_Matrices2
Scalars	floating point type	Data type of scale factor
Diagonal_Range1	integer type	Used to dimension Diagonal_Matrices1
Diagonal_Range2	integer type	Used to dimension Diagonal_Matrices2
Diagonal_Matrices1	array	An array of Elements1
Diagonal_Matrices2	array	An array of Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Elements1 * Scalars = Elements2
"/"	function	Division operator defining the operation: Elements2 / Scalars = Elements1

3.3.6.2.9.9.4 LOCAL DATA

None.

3.3.6.2.9.9.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.9.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Diagonal_Matrix_Scalar_Operations is


-----------------------------------
--begin processing for package body
-----------------------------------

begin

   --make sure instantiated diagonal matrices are of the same size
   if Diagonal_Matrices1'LENGTH /= Diagonal_Matrices2'LENGTH then
      raise Dimension_Error;
```

```
   end if;

end Diagonal_Matrix_Scalar_Operations;
```

3.3.6.2.9.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.9.8 LIMITATIONS

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the two imported vector types are not of the same length

3.3.6.2.9.9.9 LLCSC DESIGN

None.

3.3.6.2.9.9.10 UNIT DESIGN

3.3.6.2.9.9.10.1 "*" UNIT DESIGN (CATALOG #P432-0)

This function multiplies each element of a diagonal input matrix by a scale factor.

3.3.6.2.9.9.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.9.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.9.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Diagonal_Matrices1	In	Matrix to be scaled
Multiplier	Scalars	In	Scale factor

3.3.6.2.9.9.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Diagonal_Matrices2	N/A	Scaled diagonal matrix
Index1	Diagonal_Range1	N/A	Index into input matrix
Index2	Diagonal_Range2	N/A	Index into output matrix

3.3.6.2.9.9.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.9.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Matrix     : Diagonal_Matrices1;
                 Multiplier : Scalars) return Diagonal_Matrices2 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Diagonal_Matrices2;
      Index1 : Diagonal_Range1;
      Index2 : Diagonal_Range2;

-- --------------------
-- --begin function "*"-
-- --------------------
```

```
   begin

      Index1 := Diagonal_Range1'FIRST;
      Index2 := Diagonal_Range2'FIRST;
      Process:
         loop

            Answer(Index2) := Matrix(Index1) * Multiplier;

            exit Process when Index1 = Diagonal_Range1'LAST;
            Index1 := Diagonal_Range1'SUCC(Index1);
            Index2 := Diagonal_Range2'SUCC(Index2);

         end loop Process;

      return Answer;

   end "*";
```

3.3.6.2.9.9.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined in the package specification of Diagonal_Matrix_Scalar_Operations:

Name	Type	Description
Elements1	floating point type	Type of elements in Diagonal_Matrices1
Elements2	floating point type	Type of elements in Diagonal_Matrices2
Scalars	floating point type	Data type of scale factor
Diagonal_Range1	integer type	Used to dimension Diagonal_Matrices1
Diagonal_Range2	integer type	Used to dimension Diagonal_Matrices2
Diagonal_Matrices1	array	An array of Elements1
Diagonal_Matrices2	array	An array of Elements2

3.3.6.2.9.9.10.1.8 LIMITATIONS

None.

3.3.6.2.9.9.10.2 "/" UNIT DESIGN (CATALOG #P433-0)

This function divides each element of a diagonal input matrix by a scale factor.

3.3.6.2.9.9.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.9.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.9.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Diagonal_Matrices2	In	Matrix to be scaled
Divisor	Scalars	In	Scale factor

3.3.6.2.9.9.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Diagonal_Matrices1	N/A	Scaled diagonal matrix
Index1	Diagonal_Range1	N/A	Index into input matrix
Index2	Diagonal_Range2	N/A	Index into output matrix

3.3.6.2.9.9.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.9.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "/" (Matrix  : Diagonal_Matrices2;
                 Divisor : Scalars) return Diagonal_Matrices1 is
```

```
--     ----------------------
--     --declaration section-
--     ----------------------

       Answer : Diagonal_Matrices1;
       Index1 : Diagonal_Range1;
       Index2 : Diagonal_Range2;

-- ---------------------
-- --begin function "/"-
-- ---------------------

   begin

      Index1 := Diagonal_Range1'FIRST;
      Index2 := Diagonal_Range2'FIRST;
      Process:
         loop

            Answer(Index1) := Matrix(Index2) / Divisor;

            exit Process when Index1 = Diagonal_Range1'LAST;
            Index1 := Diagonal_Range1'SUCC(Index1);
            Index2 := Diagonal_Range2'SUCC(Index2);

         end loop Process;

      return Answer;

   end "/";
```

3.3.6.2.9.9.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined in the package specification of Diagonal_Matrix_Scalar_Operations:

Name	Type	Description
Elements1	floating point type	Type of elements in Diagonal_Matrices1
Elements2	floating point type	Type of elements in Diagonal_Matrices2
Scalars	floating point type	Data type of scale factor
Diagonal_Range1	integer type	Used to dimension Diagonal_Matrices1
Diagonal_Range2	integer type	Used to dimension Diagonal_Matrices2
Diagonal_Matrices1	array	An array of Elements1
Diagonal_Matrices2	array	An array of Elements2

3.3.6.2.9.9.10.2.8 LIMITATIONS

None.

3.3.6.2.9.10 MATRIX_MATRIX_MULTIPLY_UNRESTRICTED PACKAGE DESIGN (CATALOG #P439-0)

This package contains a function which multiplies an m x n matrix by an n x p matrix, returning an m x p matrix. The inner dimensions of the input matrices must be equal, the first dimensions of the left and result matrices must be the same, and the second dimensions of the right and result matrices must be the same. If any of these dimensions do not match, a Dimension_Error exception is raised.

The result of this operation is defined as follows:

```
a(i,j) := b(i,k) * c(k,j)
```

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.3.6.2.9.10.2 LOCAL ENTITIES DESIGN

Subprograms:

This package body contains a sequence of statements which are executed when it is elaborated. This code ensures that the dimensions of the instantiated matrices are as required by this part. If they are not, a Dimension_Error exception is raised.

3.3.6.2.9.10.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level of the Matrix_Matrix_Multiply_Unrestricted package:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Data type of elements in left input matrix
Right_Elements	floating point type	Data type of elements in right input matrix
Output_Elements	floating point type	Data type of elements in output matrix
Left_Col_Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part. To tailor this function to handle sparse matrices, the formal subroutines should be set up to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Left_Elements * Right_Elements := Output_Elements
"+"	function	Function defining the operation Output_Elements + Output_Elements := Output_Elements

3.3.6.2.9.10.4 LOCAL DATA

None.

3.3.6.2.9.10.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.10.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Matrix_Multiply_Unrestricted is


-----------------------------------
--begin processing for package body
-----------------------------------

begin

-- --make sure dimensions are compatible; to be compatible the following
-- --conditions must exist:
-- --must be trying to multiply: [m x n] x [n x p] := [m x p]
   if NOT (Left_Matrices'LENGTH(2)  = Right_Matrices'LENGTH(1) and      --"n's"
           Left_Matrices'LENGTH(1)  = Output_Matrices'LENGTH(1) and     --"m's"
           Right_Matrices'LENGTH(2) = Output_Matrices'LENGTH(2)) then   --"p's"

--    --dimensions are incompatible
      raise Dimension_Error;

   end if;

end Matrix_Matrix_Multiply_Unrestricted;
```

3.3.6.2.9.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.10.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the dimensions of the matrices are other than: [m x n] x [n x p] := [m x p]

3.3.6.2.9.10.9 LLCSC DESIGN

None.

3.3.6.2.9.10.10 UNIT DESIGN

3.3.6.2.9.10.10.1 "*" UNIT DESIGN (CATALOG #P440-0)

This function multiplies an m x n matrix by an n x p matrix, returning an m x p matrix.

The result of this operation is defined as follows:

```
a(i,j) := b(i,k) * c(k,j)
```

3.3.6.2.9.10.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.3.6.2.9.10.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.10.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Matrices	In	m x n matrix
Right	Right_Matrices	In	n x p matrix

3.3.6.2.9.10.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Output_Matrices	N/A	Result matrix
M_Answer	Output_Row_Indices	N/A	Index into first dimension of result matrix
M_Left	Left_Row_Indices	N/A	Index into first dimension of left input matrix
N_Left	Left_Col_Indices	N/A	Index into second dimension of left input matrix
N_Right	Right_Row_Indices	N/A	Index into first dimension of right input matrix
P_Answer	Output_Col_Indices	N/A	Index into second dimension of result matrix
P_Right	Right_Col_Indices	N/A	Index into second dimension of right input matrix

3.3.6.2.9.10.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.10.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Left  : Left_Matrices;
                 Right : Right_Matrices) return Output_Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer      : Output_Matrices;
      M_Answer    : Output_Row_Indices;
      M_Left      : Left_Row_Indices;
      N_Left      : Left_Col_Indices;
      N_Right     : Right_Row_Indices;
      P_Answer    : Output_Col_Indices;
      P_Right     : Right_Col_Indices;
```

```
-- ----------------------
-- --begin of function "*"
-- ----------------------

   begin

      M_Answer := Output_Row_Indices'FIRST;
      M_Left   := Left_Row_Indices'FIRST;
      M_Loop:
         loop

            P_Answer := Output_Col_Indices'FIRST;
            P_Right  := Right_Col_Indices'FIRST;
            P_Loop:
               loop

                  Answer(M_Answer, P_Answer) := 0.0;
                  N_Left                     := Left_Col_Indices'FIRST;
                  N_Right                    := Right_Row_Indices'FIRST;
                  N_Loop:
                     loop

                        Answer(M_Answer, P_Answer) :=
                           Answer(M_Answer, P_Answer) +
                           Left(M_Left, N_Left) * Right(N_Right, P_Right);

                        exit N_Loop when N_Left = Left_Col_Indices'LAST;
                        N_Left  := Left_Col_Indices'SUCC(N_Left);
                        N_Right := Right_Row_Indices'SUCC(N_Right);

                     end loop N_Loop;

                  exit P_Loop when P_Right = Right_Col_Indices'LAST;
                  P_Right  := Right_Col_Indices'SUCC(P_Right);
                  P_Answer := Output_Col_Indices'SUCC(P_Answer);

               end loop P_Loop;

               exit M_Loop when M_Left = Left_Row_Indices'LAST;
               M_Left   := Left_Row_Indices'SUCC(M_Left);
               M_Answer := Output_Row_Indices'SUCC(M_Answer);

         end loop M_Loop;

      return Answer;

   end "*";
```

3.3.6.2.9.10.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined in the package specification of Matrix_Matrix_Multiply_Unrestricted:

Name	Type	Description
Left_Elements	floating point type	Data type of elements in left input matrix
Right_Elements	floating point type	Data type of elements in right input matrix
Output_Elements	floating point type	Data type of elements in output matrix
Left_Col_Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_Indices	discrete type	Used to dimension second dimension of output matrix
Output_Rov_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms and task entries:

The following table describes the generic formal subprograms required by this part and defined at the package specification level of Matrix_Matrix_Multiply_-Unrestricted:

Name	Type	Description
"*"	function	Function defining the operation Left_Elements * Right_Elements := Output_Elements
"+"	function	Function defining the operation Output_Elements + Output_Elements := Output_Elements

3.3.6.2.9.10.10.1.8 LIMITATIONS

None.

3.3.6.2.9.11 MATRIX_VECTOR_MULTIPLY_UNRESTRICTED PACKAGE DESIGN (CATALOG #P434-0)

This package contains a function which multiplies an m x n matrix by an n x 1 vector producing an m x 1 vector. A DIMENSION_ERROR exception is raised if the length of the second dimension of the input matrix is not the same as the length of the input vector or if the length of the first dimension of the input matrix is not the same as the length of the output vector.

The result of this operation is defined as follows:

```
a(i) := b(i,j) * c(j)
```

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.11.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R069.

3.3.6.2.9.11.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statements which are executed when the package is elaborated. This section checks the dimensions of the instantiated arrays to ensure they are compatible for a matrix * vector := vector operation. To be compatible the following conditions must exist: Input_Matrices : m x n array Input_Vectors : n x 1 array Output_Vectors : m x 1 array If the dimensions are not compatible, a Dimension_Error exception is raised.

3.3.6.2.9.11.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level of Matrix_Vector_Multiply_Unrestricted package:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Col_Indices	discrete type	Used to dimension second dimension of input matrix
Row_Indices	discrete type	Used to dimension first dimension of input matrix
Input_Vector_Indices	discrete type	Used to dimension input vector
Output_Vector_Indices	discrete type	Used to dimension output vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Matrix_Elements * Input_Vector_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

3.3.6.2.9.11.4 LOCAL DATA

None.

3.3.6.2.9.11.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.11.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Vector_Multiply_Unrestricted is


-----------------------------------
--begin processing for package body
-----------------------------------

begin

-- --make sure dimensions are compatible; for dimensions to be compatible the following
-- --conditions must is what should be requested: [m x n] x [n x 1] = [m x 1]
   if NOT (Input_Matrices'LENGTH(2) = Input_Vectors'LENGTH and       --"n's"
           Input_Matrices'LENGTH(1) = Output_Vectors'LENGTH) then    --"m's"

--     --dimensions are incompatible
       raise Dimension_Error;

   end if;

end Matrix_Vector_Multiply_Unrestricted;
```

3.3.6.2.9.11.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.11.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the length if an operation other than the following is attempted: [m x n] x [n x 1] := [m x 1]

3.3.6.2.9.11.9 LLCSC DESIGN

None.

3.3.6.2.9.11.10 UNIT DESIGN

3.3.6.2.9.11.10.1 "*" UNIT DESIGN (CATALOG #P435-0)

This function multiplies an m x n matrix by an n x 1 vector producing an m x 1 vector.

The result of this operation is defined as follows:

```
a(i) := b(i,j) * c(j)
```

3.3.6.2.9.11.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R069.

3.3.6.2.9.11.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.11.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Input_Matrices	In	Matrix to be used as the multiplicand
Vector	Input_Vectors	In	Vector to be used as the multiplier

3.3.6.2.9.11.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Output_Vectors	Result of performing the matrix-vector multiplication
M_Answer	Output_Vector_Indices	Index into result vector
M_Matrix	Row_Indices	Index into input matrix
N_Matrix	Col_Indices	Index into input matrix
N_Vector	Input_Vector_Indices	Index into input vector

3.3.6.2.9.11.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.11.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Matrix : Input_Matrices;
                 Vector : Input_Vectors) return Output_Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Output_Vectors;
      M_Answer : Output_Vector_Indices;
      M_Matrix : Row_Indices;
      N_Matrix : Col_Indices;
      N_Vector : Input_Vector_Indices;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

      M_Answer := Output_Vector_Indices'FIRST;
      M_Matrix := Row_Indices'FIRST;
      M_Loop:
         loop

            Answer(M_Answer) := 0.0;
            N_Matrix  := Col_Indices'FIRST;
            N_Vector  := Input_Vector_Indices'FIRST;
            N_Loop:
               loop

                  Answer(M_Answer) := Answer(M_Answer) +
                              Matrix(M_Matrix, N_Matrix) * Vector(N_Vector);

                  exit N_Loop when N_Matrix = Col_Indices'LAST;
                  N_Matrix := Col_Indices'SUCC(N_Matrix);
                  N_Vector := Input_Vector_Indices'SUCC(N_Vector);
```

```
            end loop N_Loop;

            exit M_Loop when M_Matrix = Row_Indices'LAST;
            M_Matrix := Row_Indices'SUCC(M_Matrix);
            M_Answer := Output_Vector_Indices'SUCC(M_Answer);

        end loop M_Loop;

    return Answer;

end "*";
```

3.3.6.2.9.11.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table describes the generic data types required by this part and defined at the package specification level of Matrix_Vector_Multiply_-Unrestricted package:

Name	Type	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Col_Indices	discrete type	Used to dimension second dimension of input matrix
Row_Indices	discrete type	Used to dimension first dimension of input matrix
Input_Vector_Indices	discrete type	Used to dimension input vector
Output_Vector_Indices	discrete type	Used to dimension output vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms and task entries:

The following table summarizes the generic formal subroutines and required by this part and defined at the package specification level of Matrix_Vector_-Multiply_Unrestricted:

Name	Type	Description
"*"	function	Function defining the operation Matrix_Elements * Input_Vector_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

3.3.6.2.9.11.10.1.8 LIMITATIONS

None.

3.3.6.2.9.12 VECTOR_VECTOR_TRANSPOSE_MULTIPLY_UNRESTRICTED PACKAGE DESIGN (CATALOG #P442-0)

This function multiplies one input vector by the transpose of a second input vector, returning the resultant matrix. This package expects the instantiated arrays to have the following dimensions:

Left_Vectors : m x 1 array Right_Vectors : n x 1 array Matrices : m x n array

If the dimensions are not as expected, a Dimension_Error exception is raised.

The following defines the result of this operation:

```
a(i,j) := b(i) * c(j)
```

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.12.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.12.2 LOCAL ENTITIES DESIGN

Subprograms:

This package body contains a sequence of statements which checks ensure the dimensions of the instantiated vectors and arrays are required by this part. If they are not, a Dimension_Error exception is raised.

3.3.6.2.9.12.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level of the Vector_Vector_Transpose_Multiply_Unrestricted package:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Vector_Elements	floating point type	Data type of elements in left input vector
Right_Vector_Elements	floating point type	Data type of elements in right input vector
Matrix_Elements	floating point type	Data type of elements in output matrix
Left_Vector_Indices	discrete type	Used to dimension left input vector
Right_Vector_Indices	discrete type	Used to dimension right input vector
Col_Indices	discrete type	Used to dimension second dimension of output matrix
Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector
Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the multiplication operation Left_Vector_Elements * Right_Vector_Elements := Matrix_Elements

3.3.6.2.9.12.4 LOCAL DATA

None.

3.3.6.2.9.12.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.12.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Vector_Transpose_Multiply_Unrestricted is


-----------------------------------
--begin processing for package body
-----------------------------------

begin

-- --make sure dimensions are compatible;must have the following conditions:
-- --attempted operation is [m x 1] x [1 x n] := [m x n]
   if NOT (Left_Vectors'LENGTH  = Matrices'LENGTH(1) and          --"m's"
           Right_Vectors'LENGTH = Matrices'LENGTH(2)) then        --"n's"

      raise Dimension_Error;

   end if;

end Vector_Vector_Transpose_Multiply_Unrestricted;
```

3.3.6.2.9.12.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.12.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if an attempt is made to put the result of [m x 1] vector x [1 x n] vector into other than a [m x n] matrix

3.3.6.2.9.12.9 LLCSC DESIGN

None.

3.3.6.2.9.12.10 UNIT DESIGN

3.3.6.2.9.12.10.1 "*" UNIT DESIGN (CATALOG #P443-0)

This function multiplies one input vector by the transpose of a second input vector, returning the resultant matrix.

The following defines the result of this operation:

```
a(i,j) := b(i) * c(j)
```

3.3.6.2.9.12.10.1.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.12.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.12.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Vectors	In	m x 1 vector
Right	Right_Vectors	In	1 x n vector

3.3.6.2.9.12.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result matrix
M_Answer	Row_Indices	N/A	Index into first dimension of output matrix
M_Left	Left_Vector_Indices	N/A	Index into left input vector
N_Answer	Col_Indices	N/A	Index into second dimension of output matrix
N_Right	Right_Vector_Indices	N/A	Index into right input vector

3.3.6.2.9.12.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.12.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Left  : Left_Vectors ;
                 Right : Right_Vectors) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer   : Matrices;
      M_Answer : Row_Indices;
      M_Left   : Left_Vector_Indices;
      N_Answer : Col_Indices;
      N_Right  : Right_Vector_Indices;

-- ---------------------
-- --begin function "*"
-----------------------

   begin

      M_Answer := Row_Indices'FIRST;
      M_Left   := Left_Vector_Indices'FIRST;
      M_Loop:
         loop

            N_Right  := Right_Vector_Indices'FIRST;
            N_Answer := Col_Indices'FIRST;
            N_Loop:
               loop

                  Answer(M_Answer, N_Answer) := Left(M_Left) * Right(N_Right);

                  exit N_Loop when N_Right = Right_Vector_Indices'LAST;
                  N_Right  := Right_Vector_Indices'SUCC(N_Right);
                  N_Answer := Col_Indices'SUCC(N_Answer);
```

```
            end loop N_Loop;

         exit M_Loop when M_Answer = Row_Indices'LAST;
         M_Answer := Row_Indices'SUCC(M_Answer);
         M_Left   := Left_Vector_Indices'SUCC(M_Left);

   end loop M_Loop;

   return Answer;

end "*";
```

3.3.6.2.9.12.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of the Vector_Vector_Transpose_-Multiply_Unrestricted package:

Name	Type	Description
Left_Vector_Elements	floating point type	Data type of elements in left input vector
Right_Vector_Elements	floating point type	Data type of elements in right input vector
Matrix_Elements	floating point type	Data type of elements in output matrix
Left_Vector_Indices	discrete type	Used to dimension left input vector
Right_Vector_Indices	discrete type	Used to dimension right input vector
Col_Indices	discrete type	Used to dimension second dimension of output matrix
Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector
Matrices	array	Data type of output matrix

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined at the package specification level of the Vector_Vector_-Transpose_Multiply_Unrestricted package:

Name	Type	Description
"*"	function	Operator defining the multiplication operation Left_Vector_Elements * Right_Vector_Elements := Matrix_Elements

3.3.6.2.9.12.10.1.8 LIMITATIONS

None.

3.3.6.2.9.13 MATRIX_MATRIX_TRANSPOSE_MULTIPLY_UNRESTRICTED PACKAGE DESIGN (CATALOG #P445-0)

This package contains a function which multiplies one input matrix by the transpose of a second input matrix, returning the resultant matrix. The results of this operation are defined as follows:

a (i,j) := b(i,k) * c(j,k)

This package expects the instantiated arrays to have been dimensioned as follows:

Left_Matrices : m x n matrix Right_Matrices : p x n matrix Output_Matrices : m x p matrix

If the matrices have not been instantiated as expected, a Dimension_Error exception is raised.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.13.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.13.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statements which are executed when it is elaborated. This code checks to ensure the dimensions of the instantiated matrices are as required for this part. If not, a Dimension_Error exception is raised.

3.3.6.2.9.13.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level of the Matrix_Matrix_Transpose_Multiply_Unrestricted package:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input matrix
Right_Elements	floating point type	Type of elements in right input matrix
Output_Elements	floating point type	Type of elements in output matrix
Left_Col_Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator used to define the operation: Left_Elements * Right_Elements := Output_Elements

3.3.6.2.9.13.4 LOCAL DATA

None.

3.3.6.2.9.13.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.13.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Matrix_Transpose_Multiply_Unrestricted is


-----------------------------------
--begin processing for package body
-----------------------------------
begin

-- --make sure dimension are compatible
-- --need to have: [m x n] x [p x n] := [m x p]
   if NOT (Left_Matrices'LENGTH(1)  = Output_Matrices'LENGTH(1) and    --"m's"
           Left_Matrices'LENGTH(2)  = Output_Matrices'Length(2) and    --"n's"
           Right_Matrices'LENGTH(1) = Output_Matrices'LENGTH(2)) then  --"p's"

      raise Dimension_Error;

   end if;

end Matrix_Matrix_Transpose_Multiply_Unrestricted;
```

3.3.6.2.9.13.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.13.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the m's, n's, and p's of the input and output matrices are not equal; i.e., need to be doing the following operation: [m x n] x [p x n] := [m x p]

3.3.6.2.9.13.9 LLCSC DESIGN

None.

3.3.6.2.9.13.10 UNIT DESIGN

3.3.6.2.9.13.10.1 "*" UNIT DESIGN (CATALOG #P446-0)

This function multiples an m x n matrix by the transpose of a p x n matrix, returning the resultant m x p matrix.

The results of this operation are defined as follows:

```
a (i,j) := b(i,k) * c(j,k)
```

3.3.6.2.9.13.10.1.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.13.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.13.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Matrices	In	Matrix to be used as the multiplicand
Right	Right_Matrices	In	Matrix whose transpose is to be used as the multiplier

3.3.6.2.9.13.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Output_Matrices	N/A	Result matrix being calculated
M_Answer	Output_Row_Indices	N/A	Index into first dimension of result matrix
M_Left	Left_Row_Indices	N/A	Index into first dimension of left input matrix
N_Left	Left_Col_Indices	N/A	Index into second dimension of left input matrix
N_Right	Right_Col_Indices	N/A	Index into second dimension of right input matrix
P_Answer	Output_Col_Indices	N/A	Index into second dimension of result matrix
P_Right	Right_Row_Indices	N/A	Index into first dimension of right input matrix

3.3.6.2.9.13.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.13.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Left  : Left_Matrices;
                 Right : Right_Matrices) return Output_Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer   : Output_Matrices;
      M_Answer : Output_Row_Indices;
      M_Left   : Left_Row_Indices;
      N_Left   : Left_Col_Indices;
      N_Right  : Right_Col_Indices;
      P_Answer : Output_Col_Indices;
      P_Right  : Right_Row_Indices;

-- --------------------
-- --begin function "*"
-----------------------

   begin

      M_Answer := Output_Row_Indices'FIRST;
      M_Left   := Left_Row_Indices'FIRST;
      M_Loop:
         loop

            P_Answer := Output_Col_Indices'FIRST;
```

```
            P_Right  := Right_Row_Indices'FIRST;
            P_Loop:
               loop

                  Answer(M_Answer, P_Answer) := 0.0;

                  N_Left  := Left_Col_Indices'FIRST;
                  N_Right := Right_Col_Indices'FIRST;
                  N_Loop:
                     loop

                        Answer(M_Answer, P_Answer) :=
                           Answer(M_Answer, P_Answer) +
                           Left(M_Left, N_Left) * Right(P_Right, N_Right);

                        exit N_Loop when N_Left = Left_Col_Indices'LAST;
                        N_Left  := Left_Col_Indices'SUCC(N_Left);
                        N_Right := Right_Col_Indices'SUCC(N_Right);

                     end loop N_Loop;

                  exit P_Loop when P_Answer = Output_Col_Indices'LAST;
                  P_Answer := Output_Col_Indices'SUCC(P_Answer);
                  P_Right  := Right_Row_Indices'SUCC(P_Right);

               end loop P_Loop;

            exit M_Loop when M_Answer = Output_Row_Indices'LAST;
            M_Answer := Output_Row_Indices'SUCC(M_Answer);
            M_Left   := Left_Row_Indices'SUCC(M_Left);

         end loop M_Loop;

      return Answer;

   end "*";
```

3.3.6.2.9.13.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification of Matrix_Matrix_Transpose_Multiply_-Unrestricted:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input matrix
Right_Elements	floating point type	Type of elements in right input matrix
Output_Elements	floating point type	Type of elements in output matrix
Left_Col_Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms and task entries:

The following table summarizes the generic formal subroutines and task entries required by this part and defined at the package specification level of the Matrix_Matrix_Transpose_Multiply_Unrestricted package:

Name	Type	Description
"*"	function	Operator used to define the operation: Left_Elements * Right_Elements := Output_Elements

3.3.6.2.9.13.10.1.8 LIMITATIONS

None.

3.3.6.2.9.14 DOT_PRODUCT_OPERATIONS_UNRESTRICTED PACKAGE DESIGN (CATALOG #P448-0)

This package contains a function which performs a dot product operation on two m-element vectors.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.14.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R063.

3.3.6.2.9.14.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statement which are executed when the package is elaborated. This code checks to ensure the lengths of the instantiated vectors are the same.

3.3.6.2.9.14.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in this part's package specification:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input vector
Right_Elements	floating point type	Type of elements in right input vector
Result_Elements	floating point type	Data type of result of dot product
Left_Indices	discrete	Used to dimension Left_Vectors
Right_Indices	discrete	Used to dimension Right_Vectors
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Left_Elements * Right_Elements := Result_Elements

3.3.6.2.9.14.4 LOCAL DATA

None.

3.3.6.2.9.14.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.14.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Dot_Product_Operations_Unrestricted is


-----------------------------------
--begin processing for package body
-----------------------------------

begin

-- --make sure instantiated vectors are of the same length
   if Left_Vectors'LENGTH /= Right_Vectors'LENGTH then
      raise Dimension_Error;
   end if;

end Dot_Product_Operations_Unrestricted;
```

3.3.6.2.9.14.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in ancestral units:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.14.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the two input vectors is not the same

3.3.6.2.9.14.9 LLCSC DESIGN

None.

3.3.6.2.9.14.10 UNIT DESIGN

3.3.6.2.9.14.10.1 DOT_PRODUCT UNIT DESIGN (CATALOG #P449-0)

This function performs a dot product operation on two m-element vectors.

3.3.6.2.9.14.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R063.

3.3.6.2.9.14.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.14.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Vectors	in	First vector in a dot product operation
Right	Right_Vectors	in	Second vector in a dot product operation

3.3.6.2.9.14.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Result_Elements	N/A	Result of dot product operation
L_Index	Left_Indices	N/A	Index into left input vector
R_Index	Right_Indices	N/A	Index into right input vector

3.3.6.2.9.14.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.14.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function Dot_Product (Left  : Left_Vectors;
                         Right : Right_Vectors) return Result_Elements is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Result_Elements;
      L_Index : Left_Indices;
      R_Index : Right_Indices;

-- ----------------------------
-- --begin function Dot_Product-
-- ----------------------------

   begin

      Answer  := 0.0;

      L_Index := Left_Indices'FIRST;
      R_Index := Right_Indices'FIRST;
      Process:
         loop

            Answer := Answer + Left(L_Index) * Right(R_Index);

            exit Process when L_Index = Left_Indices'LAST;
            L_Index := Left_Indices'SUCC(L_Index);
            R_Index := Right_Indices'SUCC(R_Index);

         end loop Process;

      return Answer;

   end Dot_Product;
```

3.3.6.2.9.14.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of the Dot_Product_Operations_-Unrestricted package:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input vector
Right_Elements	floating point type	Type of elements in right input vector
Result_Elements	floating point type	Data type of result of dot product
Left_Indices	discrete	Used to dimension Left_Vectors
Right_Indices	discrete	Used to dimension Right_Vectors
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector

3.3.6.2.9.14.10.1.8 LIMITATIONS

None.

3.3.6.2.9.15 DIAGONAL_FULL_MATRIX_ADD_UNRESTRICTED PACKAGE DESIGN (CATALOG #P451-0)

This package contains a function adds a diagonal matrix to a full matrix by adding the individual elements of the input matrices.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.15.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.15.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains code which is executed when the package is elaborated. This code checks to make sure the dimensions of the instantiated arrays are compatible. The diagonal matrix should have m elements, and both of the full matrices should be m x m arrays. If these conditions are not met, a

Dimension_Error exception is raised.

3.3.6.2.9.15.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification of Diagonal_Full_Matrix_Add_Unrestricted:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Type of elements in input and output arrays
Diagonal_Range	integer type	Used to dimension Diagonal_Matrices
Full_Input_Col_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Input_Row_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Output_Col_Indices	discrete	Used to dimension Full_Output_ matrices
Full_Output_Row_Indices	discrete	Used to dimension Full_Output_ matrices
Diagonal_Matrices	array	Data type of diagonal input matrix
Full_Input_Matrices	array	Data type of full input matrix
Full_Output_Matrices	array	Data type of full output matrix

3.3.6.2.9.15.4 LOCAL DATA

None.

3.3.6.2.9.15.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.15.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Diagonal_Full_Matrix_Add_Unrestricted is


------------------------------
--begin package body processing
```

```
------------------------------
begin

-- --make sure square matrices of the same size have been instantiated
   if not (Diagonal_Matrices'LENGTH       = Full_Input_Matrices'LENGTH(1)  and
           Full_Input_Matrices'LENGTH(1)  = Full_Input_Matrices'LENGTH(2)  and
           Full_Input_Matrices'LENGTH(1)  = Full_Output_Matrices'LENGTH(1) and
           Full_Output_Matrices'LENGTH(1) = Full_Output_Matrices'LENGTH(2)) then

     raise Dimension_Error;

   end if;

end Diagonal_Full_Matrix_Add_Unrestricted;
```

3.3.6.2.9.15.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.15.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the matrix indices are not equal to each other and other the length of the diagonal matrix

3.3.6.2.9.15.9 LLCSC DESIGN

None.

3.3.6.2.9.15.10 UNIT DESIGN

3.3.6.2.9.15.10.1 "+" UNIT DESIGN (CATALOG #P452-0)

This function adds an m-element diagonal matrix to an m x m matrix, returning the resultant m x m matrix.

3.3.6.2.9.15.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.15.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.15.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
D_Matrix	Diagonal_Matrices	In	Input diagonal matrix
F_Matrix	Full_Input_Matrices	In	Input full matrix to be added to the diagonal matrix

3.3.6.2.9.15.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Full_Output_Matrices	N/A	Resultant matrix
A_Col_Index	Full_Output_Col_Indices	N/A	Index into second dimension of Answer matrix
A_Col_Marker	Full_Output_Col_Indices	N/A	Marks a column in Answer matrix which contains the diagonal element in row A_Row_Index
A_Row_Index	Full_Output_Row_Indices	N/A	Index into first dimension of Answer matrix
D_Index	Diagonal_Range	N/A	Index into diagonal matrix
F_Col_Index	Full_Input_Col_Indices	N/A	Index into second dimension of F_Matrix
F_Row_Index	Full_Input_Row_Indices	N/A	Index into first dimension of F_Matrix

3.3.6.2.9.15.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.15.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (D_Matrix : Diagonal_Matrices;
                 F_Matrix : Full_Input_Matrices) return Full_Output_Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer       : Full_Output_Matrices;
      A_Col_Index  : Full_Output_Col_Indices;
      A_Col_Marker : Full_Output_Col_Indices;
      A_Row_Index  : Full_Output_Row_Indices;
      D_Index      : Diagonal_Range;
      F_Col_Index  : Full_Input_Col_Indices;
      F_Row_Index  : Full_Input_Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --first assign a row full of values, then add in diagonal element

      A_Col_Marker := Full_Output_Col_Indices'FIRST;
      A_Row_Index  := Full_Output_Row_Indices'FIRST;
      D_Index      := Diagonal_Range'FIRST;
      F_Row_Index  := Full_Input_Row_Indices'FIRST;
```

```
      Add_Loop:
         loop

            A_Col_Index := Full_Output_Col_Indices'FIRST;
            F_Col_Index := Full_Input_Col_Indices'FIRST;
            Assign_Loop:
               loop

                  Answer(A_Row_Index, A_Col_Index) :=
                     F_Matrix(F_Row_Index, F_Col_Index);

                  exit Assign_Loop
                     when A_Col_Index = Full_Output_Col_Indices'LAST;
                  A_Col_Index := Full_Output_Col_Indices'SUCC(A_Col_Index);
                  F_Col_Index := Full_Input_Col_Indices'SUCC(F_Col_Index);

               end loop Assign_Loop;

            Answer(A_Row_Index, A_Col_Marker) :=
               Answer(A_Row_Index, A_Col_Marker) + D_Matrix(D_Index);

            exit Add_Loop when D_Index = Diagonal_Range'LAST;
            A_Col_Marker := Full_Output_Col_Indices'SUCC(A_Col_Marker);
            A_Row_Index  := Full_Output_Row_Indices'SUCC(A_Row_Index);
            D_Index      := D_Index + 1;
            F_Row_Index  := Full_Input_Row_Indices'SUCC(F_Row_Index);

         end loop Add_Loop;

      return Answer;

   end "+";
```

3.3.6.2.9.15.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Diagonal_Full_Matrix_Add_-Unrestricted:

Name	Type	Description
Elements	floating point type	Type of elements in input and output arrays
Diagonal_Range	integer type	Used to dimension Diagonal_Matrices
Full_Input_Col_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Input_Row_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Output_Col_Indices	discrete	Used to dimension Full_Output_ matrices
Full_Output_Row_Indices	discrete	Used to dimension Full_Output_ matrices
Diagonal_Matrices	array	Data type of diagonal input matrix
Full_Input_Matrices	array	Data type of full input matrix
Full_Output_Matrices	array	Data type of full output matrix

3.3.6.2.9.15.10.1.8 LIMITATIONS

None.

3.3.6.2.9.16 VECTOR_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P342-0)

This package contains functions which provide a set of standard vector operations. The operations provided are addition, subtraction, and dot product of like vectors, along with a vector length operation.

The vectors operated upon by parts in this part are constrained arrays.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.16.1 REQUIREMENTS ALLOCATION

The following table describes the allowing of requirements to this part:

Name	Requirements Allocation
Dot_Product	R063
Vector_Length	R104
"+"	R061
"-"	R062

3.3.6.2.9.16.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were defined at the package specification level:

Data types:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_Squared	floating point type	Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

Subprograms:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared
SqRt	function	Square root function taking an object of type Vector_Elements_Squared and returning an object of type Vector_Elements

3.3.6.2.9.16.4 LOCAL DATA

Data types:

The following table summarizes the types defined in this part's specification:

Name	Range	Description
Vectors	N/A	Constrained, one-dimensional array of elements

3.3.6.2.9.16.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Operations_Constrained is

end Vector_Operations_Constrained;
```

3.3.6.2.9.16.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.16.8 LIMITATIONS

None.

3.3.6.2.9.16.9 LLCSC DESIGN

None.

3.3.6.2.9.16.10 UNIT DESIGN

3.3.6.2.9.16.10.1 "+" (VECTOR + VECTORS := VECTORS) UNIT DESIGN (CATALOG #P343-0)

This function adds two vectors by adding each of the individual elements in the input vector, returning the resultant vector.

3.3.6.2.9.16.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R061.

3.3.6.2.9.16.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	One of the vectors to be added
Right	Vectors	In	Second vector to be added

3.3.6.2.9.16.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Vector being calculated and returned

3.3.6.2.9.16.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Vectors;
                 Right : Vectors) return Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Vectors;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Process:
         for Index in Indices loop

            Answer(Index) := Left(Index) + Right(Index);

         end loop Process;

      return Answer;

   end "+";
```

3.3.6.2.9.16.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and are defined in the package specification for Vector_Operations_Constrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Constrained:

Name	Range	Description
Vectors	N/A	Constrained, one-dimensional array of elements

3.3.6.2.9.16.10.1.8 LIMITATIONS

None.

3.3.6.2.9.16.10.2 "-" (VECTORS - VECTORS := VECTORS) UNIT DESIGN (CATALOG #P344-0)

This part subtracts one vector from another by subtracting the individual elements of each input vector, returning the resultant vector.

3.3.6.2.9.16.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R062.

3.3.6.2.9.16.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	Vector to act as the minuend
Right	Vectors	In	Vector to act as the subtrahend

3.3.6.2.9.16.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Vector being calculated and returned

3.3.6.2.9.16.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Vectors;
                 Right : Vectors) return Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Vectors;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Process:
         for Index in Indices loop

            Answer(Index) := Left(Index) - Right(Index);

         end loop Process;
```

```
      return Answer;

   end "-";
```

3.3.6.2.9.16.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector_Operations_Constrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Constrained:

Name	Range	Description
Vectors	N/A	Constrained, one-dimensional array of elements

3.3.6.2.9.16.10.2.8 LIMITATIONS

None.

3.3.6.2.9.16.10.3 VECTOR_LENGTH UNIT DESIGN (CATALOG #P345-0)

This function calculates the length of a vector, returning the result. The length of a vector is defined as:

```
    a := Sqrt(sum b(i)**2)
```

3.3.6.2.9.16.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R104.

3.3.6.2.9.16.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Vectors	In	Vector for which a length is desired

3.3.6.2.9.16.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Temp	Vector_Elements_Squared	Used for intermediate calculations

3.3.6.2.9.16.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   function Vector_Length (Input : Vectors) return Vector_Elements is

--    ----------------------
--    --declaration section-
--    ----------------------

      Temp   : Vector_Elements_Squared;

-- ------------------------------
-- --begin function Vector_Length
-- ------------------------------

   begin

      Temp := 0.0;

      Process:
```

```
         for Index in Indices loop
            Temp := Temp +
                    Input(Index) * Input(Index);
         end loop Process;

      return SqRt(Temp);

   end Vector_Length;
```

3.3.6.2.9.16.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector_Operations_Constrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_ Squared	floating point type	Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Constrained:

Name	Range	Description
Vectors	N/A	Constrained, one-dimensional array of elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

Subprograms:

The following table summarizes the generic subroutines available to this part and defined at the package specification level for Vector_Operations_-Constrained:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared
SqRt	function	Square root function taking an object of type Vector_Elements_Squared and returning an object of type Vector_Elements

3.3.6.2.9.16.10.3.8 LIMITATIONS

None.

3.3.6.2.9.16.10.4 DOT PRODUCT UNIT DESIGN (CATALOG #P346-0)

This function calculates the dot product of two vectors by keeping a running sum of the product of each element of the input vectors.

3.3.6.2.9.16.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R063.

3.3.6.2.9.16.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	First vector to be used in the dot product operation
Right	Vectors	In	Second vector to be used in the dot product operation

3.3.6.2.9.16.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Vector_Elements_Squared	N/A	Result of dot product operation

3.3.6.2.9.16.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function Dot_Product (Left  : Vectors;
                         Right : Vectors) return Vector_Elements_Squared is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Vector_Elements_Squared;

-- ----------------------------
-- --begin function Dot_Product
-- ----------------------------

   begin

      Answer := 0.0;

      Process:
         for Index in Indices loop

            Answer := Answer + Left(Index) * Right(Index);

         end loop Process;

      return Answer;

   end Dot_Product;
```

3.3.6.2.9.16.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector_Operations_Constrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_Squared	floating point type	Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Constrained:

Name	Range	Description
Vectors	N/A	Constrained, one-dimensional array of elements

Subprograms:

The following table summarizes the generic subroutines available to this part and defined at the package specification level for Vector_Operations:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared

3.3.6.2.9.16.10.4.8 LIMITATIONS

None.

3.3.6.2.9.17 MATRIX_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P355-0)

This package contains subroutines which provide a set of standard operations on matrices of like types.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.17.1 REQUIREMENTS ALLOCATION

This following illustrates the allocation of requirements to the units in this package.

Name	Requirements Allocation
"+" (matrices + matrices)	R079
"-" (matrices - matrices)	R080
"+" (matrices + elements)	R075
"-" (matrices - elements)	R076
Set_to_Identity_Matrix	R155
Set_to_Zero_Matrix	R156

3.3.6.2.9.17.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level:

Data types:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

3.3.6.2.9.17.4 LOCAL DATA

Data types:

The following data type was previously defined at the package specification level:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Operations_Constrained is

end Matrix_Operations_Constrained;
```

3.3.6.2.9.17.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.17.8 LIMITATIONS

None.

3.3.6.2.9.17.9 LLCSC DESIGN

None.

3.3.6.2.9.17.10 UNIT DESIGN

3.3.6.2.9.17.10.1 "+" (MATRICES + MATRICES := MATRICES) UNIT DESIGN (CATALOG #P356-0)

This function adds two matrices by adding the individual elements of each input matrix, returning the resultant matrix.

3.3.6.2.9.17.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R079.

3.3.6.2.9.17.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	First matrix to be added
Right	Matrices	In	Second matrix to be added

3.3.6.2.9.17.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Result of adding the two input matrices

3.3.6.2.9.17.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for Col in Col_Indices loop

                  Answer(Row, Col) := Left(Row, Col) +
                                      Right(Row, Col);

               end loop Col_Loop;

         end loop Row_Loop;
```

```
      return Answer;

   end "+";
```

3.3.6.2.9.17.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.1.8 LIMITATIONS

None.

3.3.6.2.9.17.10.2 "-" (MATRICES - MATRICES := MATRICES) UNIT DESIGN (CATALOG #P357-0)

This function subtracts one matrix from another by subtracting the individual elements of the input matrices, returning the resultant matrix.

3.3.6.2.9.17.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R080.

3.3.6.2.9.17.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Matrix to act as the minuend
Right	Matrices	In	Matrix to be used as the subtrahend

3.3.6.2.9.17.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Result of adding the two input matrices

3.3.6.2.9.17.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Matrices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Row_Loop:
```

```
        for Row in Row_Indices loop

           Col_Loop:
              for Col in Col_Indices loop

                 Answer(Row, Col) := Left(Row, Col) -
                                     Right(Row, Col);

              end loop Col_Loop;

        end loop Row_Loop;

     return Answer;

  end "-";
```

3.3.6.2.9.17.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification for Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.2.8 LIMITATIONS

None.

3.3.6.2.9.17.10.3 "+" (MATRICES + ELEMENTS := MATRICES) UNIT DESIGN (CATALOG #P358-0)

This function calculates a scaled matrix by adding a scale factor to each element of an input matrix, returning the resultant matrix.

3.3.6.2.9.17.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R075.

3.3.6.2.9.17.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be scaled
Addend	Elements	In	Scale factor

3.3.6.2.9.17.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Scaled matrix

3.3.6.2.9.17.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Matrix : Matrices;
                 Addend : Elements) return Matrices is

--     ---------------------
```

```
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop
            Col_Loop:
               for Col in Col_Indices loop
                  Answer(Row, Col) := Matrix(Row, Col) + Addend;
               end loop Col_Loop;
         end loop Row_Loop;

      return Answer;

   end "+
```

3.3.6.2.9.17.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.3.8 LIMITATIONS

None.

3.3.6.2.9.17.10.4 "-" (MATRICES - ELEMENTS := MATRICES) UNIT DESIGN (CATALOG #P359-0)

This function calculates a scaled matrix by subtracting a scale factor from each element of an input matrix, returning the resultant matrix.

3.3.6.2.9.17.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R076.

3.3.6.2.9.17.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be scaled
Subtrahend	Elements	In	Scale factor

3.3.6.2.9.17.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Scaled matrix

3.3.6.2.9.17.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Matrix     : Matrices;
                 Subtrahend : Elements) return Matrices is

--     ----------------- ----
--     --declaration section-
--     ----------------------

       Answer : Matrices;

-- ---------- ----------
-- --begin function "-"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop
            Col_Loop:
               for Col in Col_Indices loop
                  Answer(Row, Col) := Matrix(Row, Col) - Subtrahend;
               end loop Col_Loop;
         end loop Row_Loop;

      return Answer;

   end "-";
```

3.3.6.2.9.17.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.4.8 LIMITATIONS

None.

3.3.6.2.9.17.10.5 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P360-0)

This procedure turns an input matrix into an identity matrix. An identity matrix is one in which the diagonal elements equal 1.0 and all other elements equal 0.0. The input matrix must be a square matrix, but the ranges of the individual dimensions do not have to be the same.

3.3.6.2.9.17.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R155.

3.3.6.2.9.17.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	Out	Matrix to be made into an identity matrix

3.3.6.2.9.17.10.5.4 LOCAL DATA

Data objects:

The following describes the data objects maintained local to this part:

Name	Type	Description
Col	Col_Indices	Index into second dimension of matrix
Row	Row_Indices	Index into first dimension of matrix

3.3.6.2.9.17.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--    ----------------------
--    --declaration section
--    ----------------------

      Col : Col_Indices;
      Row : Row_Indices;

-- ---------------------------------------
-- --begin procedure Set_To_Identity_Matrix
-- ---------------------------------------

   begin

--    --make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

         Matrix := (others => (others => 0.0));

         Col := Col_Indices'FIRST;
         Row := Row_Indices'FIRST;
         Row_Loop:
            loop

--             --set diagonal element equal to 1
               Matrix(Row, Col) := 1.0;

               exit when Row = Row_Indices'LAST;
               Col := Col_Indices'SUCC(Col);
               Row := Row_Indices'SUCC(Row);

            end loop Row_Loop;

      else

--       --do not have a square matrix
         raise Dimension_Error;

      end if;
```

```
   end Set_To_Identity_Matrix;
```

3.3.6.2.9.17.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.17.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	Description
Dimension_Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.17.10.6 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P361-0)

This procedure zeros out all elements of an input matrix.

3.3.6.2.9.17.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R156.

3.3.6.2.9.17.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	Out	Matrix to be zeroed out

3.3.6.2.9.17.10.6.4 LOCAL DATA

None.

3.3.6.2.9.17.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
begin
   Matrix := (others => (others => 0.0));
end Set_To_Zero_Matrix;
```

3.3.6.2.9.17.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification for Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.6.8 LIMITATIONS

None.

3.3.6.2.9.18 DYNAMICALLY_SPARSE_MATRIX_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P369-0)

This package defines a dynamically sparse matrix and operations on it. All elements of the matrix are stored, but most of the elements are expected to be 0. Which elements are zero does not have to remain the same. See decomposition section for the operations provided.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.18.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described at the package specification level:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

3.3.6.2.9.18.4 LOCAL DATA

Data types:

The following data types were previously defined at the package specification level:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.18.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Dynamically_Sparse_Matrix_Operations_Constrained is

end Dynamically_Sparse_Matrix_Operations_Constrained;
```

3.3.6.2.9.18.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.18.8 LIMITATIONS

None.

3.3.6.2.9.18.9 LLCSC DESIGN

None.

3.3.6.2.9.18.10 UNIT DESIGN

3.3.6.2.9.18.10.1 SET_TO_IDENTITY_MATRIX UNIT DESIGN (CATALOG #P370-0)

This procedure sets a square input matrix to an identity matrix. An identity matrix is one where the diagonal elements all equal 1.0, with the remaining elements equaling 0.0.

3.3.6.2.9.18.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix being made into an identity matrix

3.3.6.2.9.18.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Col	Col_Indices	Index into second dimension of input matrix
Row	Row_Indices	Index into first dimension of input matrix

3.3.6.2.9.18.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--    ---------------------
--    --declaration section
--    ---------------------

      Col : Col_Indices;
      Row : Row_Indices;

-- ----------------------------------------
-- --begin procedure Set_to_Identity_Matrix
-- ----------------------------------------

   begin

--    --make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

         Matrix := (others => (others => 0.0));

         Col := Col_Indices'FIRST;
         Row := Row_Indices'FIRST;
         Row_Loop:
            loop

--             --set diagonal element equal to 1.0
               Matrix(Row, Col) := 1.0;

               exit when Row = Row_Indices'LAST;
               Col := Col_Indices'SUCC(Col);
               Row := Row_Indices'SUCC(Row);

            end loop Row_Loop;

      else

         raise Dimension_Error;

      end if;
```

```
   end Set_to_Identity_Matrix;
```

3.3.6.2.9.18.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.18.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.18.10.2 SET_TO_ZERO_MATRIX UNIT DESIGN (CATALOG #P371-0)

This procedure sets all elements of an input matrix to zero.

3.3.6.2.9.18.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be zeroed out

3.3.6.2.9.18.10.2.4 LOCAL DATA

None.

3.3.6.2.9.18.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
begin
   Matrix := (others => (others => 0.0));
end Set_to_Zero_Matrix;
```

3.3.6.2.9.18.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.18.10.2.8 LIMITATIONS

None.

3.3.6.2.9.18.10.3 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P372-0)

This function takes a square input matrix and adds it to an identity matrix by adding 1.0 to all diagonal elements of the input matrix.

3.3.6.2.9.18.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Matrix to which is added an identity matrix

3.3.6.2.9.18.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding an identity matrix to the input matrix
Col	Col_Indices	N/A	Column index
Row	Row_Indices	N/A	Row index

3.3.6.2.9.18.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   function Add_to_Identity (Input : Matrices) return Matrices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer : Matrices;
       Col    : Col_Indices;
       Row    : Row_Indices;

-- -------------------------------
-- --begin function Add_to_Identity
-- -------------------------------

   begin

--     --make sure input is a square matrix
       if Input'LENGTH(1) = Input'LENGTH(2) then
```

```
         Answer := Input;

--       --add "identity" values to diagonal elements
         Col := Col_Indices'FIRST;
         Row := Row_Indices'FIRST;
         Row_Loop:
            loop

               if Answer(Row, Col) /= 0.0 then
                  Answer(Row, Col) := Answer(Row, Col) + 1.0;
               else
                  Answer(Row, Col) := 1.0;
               end if;

               exit when Row = Row_Indices'LAST;
               Col := Col_Indices'SUCC(Col);
               Row := Row_Indices'SUCC(Row);

            end loop Row_Loop;

      else

         raise Dimension_Error;

      end if;

      return Answer;

   end Add_to_Identity;
```

3.3.6.2.9.18.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.18.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.18.10.4 SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P373-0)

This function subtracts a square input matrix from an identity matrix by negating all elements of an input matrix and then adding 1.0 to the elements on the diagonal.

3.3.6.2.9.18.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Square matrix to be subtracted from an identity matrix

3.3.6.2.9.18.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of subtracting input matrix from an identity matrix
Col	Col_Indices	N/A	Column index
Row	Row_Indices	N/A	Row index

3.3.6.2.9.18.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function Subtract_from_Identity (Input : Matrices) return Matrices is

--     --------------------
--     --declaration section
--     --------------------

       Answer  : Matrices;
       Col     : Col_Indices;
       Row     : Row_Indices;

-- ----------------------------------------
-- --begin procedure Subtract_From_Identity
-- ----------------------------------------

   begin

--     --make sure input is a square matrix
       if Input'LENGTH(1) = Input'LENGTH(2) then

          Col := Col_Indices'FIRST;
          Row := Row_Indices'FIRST;
          Row_Loop:
```

```
            loop

               Col_Loop:
                  for Temp_Col in Col_Indices loop
                     if Input(Row,Temp_Col) /= 0.0 then
                        Answer(Row,Temp_Col) := - Input(Row,Temp_Col);
                     else
                        Answer(Row,Temp_Col) := 0.0;
                     end if;
                  end loop Col_Loop;

               if Answer(Row, Col) /= 0.0 then
                  Answer(Row, Col) := Answer(Row, Col) + 1.0;
               else
                  Answer(Row, Col) := 1.0;
               end if;

               exit when Row = Row_Indices'LAST;
               Col := Col_Indices'SUCC(Col);
               Row    Row_Indices'SUCC(Row);

            end loop Row_Loop;

         else

            raise Dimension_Error;

         end if;

         return Answer;

      end Subtract_From_Identity;
```

3.3.6.2.9.18.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.18.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.18.10.5 "+" UNIT DESIGN (CATALOG #P374-0)

This function adds two sparse m x n matrices, by adding the individual elements of the input matrices taking advantage of the fact that most of the elements of both matrices equal 0.

3.3.6.2.9.18.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Sparse matrix to be added
Right	Matrices	In	Sparse matrix to be added

3.3.6.2.9.18.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding two input matrices

3.3.6.2.9.18.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for Col in Col_Indices loop

                  if Left(Row, Col) = 0.0 then
                     if Right(Row, Col) = 0.0 then
                        Answer(Row, Col) := 0.0;
                     else
```

```
                        Answer(Row, Col) := Right(Row, Col);
                     end if;
                  elsif Right(Row, Col) = 0.0 then
                     Answer(Row, Col) := Left(Row, Col);
                  else
                     Answer(Row, Col) := Left(Row, Col) +
                                         Right(Row, Col);
                  end if;

               end loop Col_Loop;

            end loop Row_Loop;

         return Answer;

      end "+";
```

3.3.6.2.9.18.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.18.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if both matrices are not m x n matrices

3.3.6.2.9.18.10.6 "-" UNIT DESIGN (CATALOG #P375-0)

This function subtracts two sparse m x n matrices by subtracting the individual elements of the input matrices, taking advantage of the fact that most of the elements of both matrices equal 0.

3.3.6.2.9.18.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Sparse matrix to be treated as the minuend
Right	Matrices	In	Sparse matrix to be treated as the subtrahend

3.3.6.2.9.18.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of subtracting two input matrices

3.3.6.2.9.18.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section
--    ----------------------

      Answer : Matrices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for Col in Col_Indices loop

                  if Left(Row, Col) = 0.0 then
                     if Right(Row, Col) = 0.0 then
                        Answer(Row, Col) := 0.0;
                     else
                        Answer(Row, Col) := - Right(Row, Col);
                     end if;
                  elsif Right(Row, Col) = 0.0 then
                     Answer(Row, Col) := Left(Row, Col);
                  else
                     Answer(Row, Col) := Left(Row, Col) -
                                         Right(Row, Col);
                  end if;

               end loop Col_Loop;

         end loop Row_Loop;

      return Answer;

   end "-";
```

3.3.6.2.9.18.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.18.10.6.8 LIMITATIONS

None.

3.3.6.2.9.19 SYMMETRIC_FULL_STORAGE_MATRIX_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P398-0)

This package exports operations on a symmetric full storage matrix.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.19.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.19.2 LOCAL ENTITIES DESIGN

Subprograms:

There exists a sequence of statements at the end of this package body which are executed when this part is elaborated. The code checks to ensure a square matrix has been instantiated. If not, a Dimension_Error exception is raised.

3.3.6.2.9.19.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level:

Data types:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

3.3.6.2.9.19.4 LOCAL DATA

Data types:

The following types are previously defined in this part's package specification:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Symmetric_Full_Storage_Matrix_Operations_Constrained is


-------------------------------------------
--processing for Symmetric_Full_Storage_
--Matrix_Operations_Constrained package body
-------------------------------------------

begin

   if Matrices'LENGTH(1) /= Matrices'LENGTH(2) then
      raise Dimension_Error;
   end if;

end Symmetric_Full_Storage_Matrix_Operations_Constrained;
```

3.3.6.2.9.19.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following elements are required by this part and defined elsewhere in the TLCSC.

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra.

Name	Description
Dimension_Error	Raised by a routine or package when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.19.8 LIMITATIONS

None.

3.3.6.2.9.19.9 LLCSC DESIGN

None.

3.3.6.2.9.19.10 UNIT DESIGN

3.3.6.2.9.19.10.1 CHANGE_ELEMENT UNIT DESIGN (CATALOG #P399-0)

This procedure changes the indicated element of a symmetric matrix, along with its symmetric counterpart.

3.3.6.2.9.19.10.1.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Value	Elements	In	New value to be placed in the matrix
Row	Row_Indices	In	Row in which the value belongs
Col	Col_Indices	In	Column in which the value belongs
Matrix	Matrices	In/Out	Matrix being updated

3.3.6.2.9.19.10.1.4 LOCAL DATA

None.

3.3.6.2.9.19.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Change_Element (New_Value : in     Elements;
                             Row       : in     Row_Indices;
                             Col       : in     Col_Indices;
                             Matrix    : in out Matrices) is

--    ----------------------
--    --declaration section-
--    ----------------------
```

```
      S_Col : Col_Indices;
      S_Row : Row_Indices;

-- --------------------------------
-- --begin procedure Change_Element-
-- --------------------------------

   begin

      S_Col := Col_Indices'VAL(Row_Indices'POS(Row) -
                               Row_Indices'POS(Row_Indices'FIRST) +
                               Col_Indices'POS(Col_Indices'FIRST));

      S_Row := Row_Indices'VAL(Col_Indices'POS(Col) -
                               Col_Indices'POS(Col_Indices'FIRST) +
                               Row_Indices'POS(Row_Indices'FIRST));

      Matrix(Row, Col) := New_Value;
      Matrix(S_Row, S_Col) := New_Value;

   end Change_Element;
```

3.3.6.2.9.19.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.1.8 LIMITATIONS

None.

3.3.6.2.9.19.10.2 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P400-0)

This procedure turns an input matrix into an identity matrix. An identity matrix is one where all elements equal 0.0, except those on the diagonal which equal 1.0. The input matrix must be a square matrix.

3.3.6.2.9.19.10.2.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	Out	Matrix to be made into an identity matrix

3.3.6.2.9.19.10.2.4 LOCAL DATA

Data objects:

The following table describes the local data maintained by this part:

Name	Type	Value	Description
Col	Col_Indices	N/A	Index into second dimension of matrix
Row	Row_Indices	N/A	Index into first dimension of matrix

3.3.6.2.9.19.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--    ---------------------
--    --declaration section
--    ---------------------

      Col : Col_Indices;
      Row : Row_Indices;

-- ----------------------------------------
-- --begin procedure Set_to_Identity_Matrix
-- ----------------------------------------

   begin

      Matrix := (others => (others => 0.0));

      Col := Col_Indices'FIRST;
      Row := Row_Indices'FIRST;
      Row_Loop:
         loop

--          --set diagonal element equal to
            Matrix(Row, Col) := 1.0;

            exit when Row = Row_Indices'LAST;
            Col := Col_Indices'SUCC(Col);
            Row := Row_Indices'SUCC(Row);

         end loop Row_Loop;

   end Set_To_Identity_Matrix;
```

3.3.6.2.9.19.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.2.8 LIMITATIONS

None.

3.3.6.2.9.19.10.3 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P401-0)

This procedure zeros out all elements of an input matrix.

3.3.6.2.9.19.10.3.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	Out	Matrix to be zeroed out

3.3.6.2.9.19.10.3.4 LOCAL DATA

None.

3.3.6.2.9.19.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

begin

   Matrix := (others => (others => 0.0));

end Set_To_Zero_Matrix;
```

3.3.6.2.9.19.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.3.8 LIMITATIONS

None.

3.3.6.2.9.19.10.4 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P402-0)

This function adds an input matrix to an identity matrix, returning the resultant matrix. The addition is performed by adding 1.0 to each diagonal element of the input matrix.

3.3.6.2.9.19.10.4.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Matrix to be added to an identity matrix

3.3.6.2.9.19.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding identity matrix to input matrix
Col	Col_Indices	N/A	Index into second dimension of matrices
Row	Row_Indices	N/A	Index into first dimension of matrices

3.3.6.2.9.19.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function Add_to_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer     : Matrices;
      Col        : Col_Indices;
      Row        : Row_Indices;

-- --------------------------------
-- --begin function Add_to_Identity
-- --------------------------------

   begin

      Answer := Input;

      Col := Col_Indices'FIRST;
      Row := Row_Indices'FIRST;
      Access_Diagonal_Elements:
         loop

            Answer(Row,Col) := Answer(Row,Col) + 1.0;

            exit when Row = Row_Indices'LAST;
            Col := Col_Indices'SUCC(Col);
            Row := Row_Indices'SUCC(Row);

         end loop Access_Diagonal_Elements;

      return Answer;

   end Add_to_Identity;
```

3.3.6.2.9.19.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.4.8 LIMITATIONS

None.

3.3.6.2.9.19.10.5 SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P403-0)

This function subtracts an input matrix from an identity matrix, returning the resultant matrix.

3.3.6.2.9.19.10.5.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Input	Matrices	In	Matrix to be subtracted from an identity matrix

3.3.6.2.9.19.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding input matrices
Row	Row_Indices	N/A	Row index into matrix
S_Col	Col_Indices	N/A	"Symmetric" column index into matrix
S_Row	Row_Indices	N/A	"Symmetric" row index into matrix; i.e., A(row,col) := A(s_row,s_col)

3.3.6.2.9.19.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
   function Subtract_from_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer    : Matrices;
      Row       : Row_Indices;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- --------- ----------------------------
-- --begin function Subtract_from_Identity
-- ---------------------------------------

   begin

--    --handle first diagonal element

      Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
         1.0 - Input(Row_Indices'FIRST, Col_Indices'FIRST);

--    --will subtract the remaining of the input matrix from an identity matrix
--    --by doing the following:
--    --  o subtracting the nondiagonal elements in the bottom half of the
--    --      matrix from 0.0,
--    --  o assigning values obtained in the bottom half of the matrix to the
--    --      symmetric elements in the top half of the matrix, and then
--    --  o subtracting the diagonal elements from 1.0

--    --S_Col will go across the columns as Row goes down the rows to keep
```

```
--     --track of the column containing the diagonal element
       S_Col := Col_Indices'SUCC(Col_Indices'FIRST);
       Row   := Row_Indices'SUCC(Row_Indices'FIRST);
       Do_Every_Row_Except_First:
          loop

--           --S_Row will go down the rows as Col goes across the columns
             S_Row := Row_Indices'FIRST;
             Subtract_Nondiagonal_Elements_From_Zero:
                for Col in Col_Indices'FIRST ..
                           Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop

                   Answer(Row,Col) := - Input(Row,Col);

                   Answer(S_Row,S_Col) := Answer(Row,Col);

                   S_Row     := Row_Indices'SUCC(S_Row);

                end loop Subtract_Nondiagonal_Elements_From_Zero;

--           --subtract diagonal element from 1.0
             Answer(Row, S_Col) := 1.0 - Input(Row, S_Col);

             exit when Row = Row_Indices'LAST;
             S_Col := Col_Indices'SUCC(S_Col);
             Row   := Row_Indices'SUCC(Row);

          end loop Do_Every_Row_Except_First;

       return Answer;

    end Subtract_from_Identity;
```

3.3.6.2.9.19.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if input matrix is not a square matrix

3.3.6.2.9.19.10.6 "+" UNIT DESIGN (CATALOG #P404-0)

This function adds two symmetric matrices by adding the individual elements of the input matrices, taking advantage of their symmetricity.

3.3.6.2.9.19.10.6.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	First matrix to be added
Right	Matrices	In	Second matrix to be added

3.3.6.2.9.19.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding two input matrices
Row	Row_Indices	N/A	Index into first dimension of matrix
S_Col	Col_Indices	N/A	Used to keep track of column containing diagonal element for the current row
S_Row	Row_Indices	N/A	When used with S_Col, marks the symmetric counterpart to the element being referenced in the bottom half of the array; i.e., A(row,col) = A(s_row,s_col)

3.3.6.2.9.19.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    --------------------
--    --declaration section
--    --------------------

      Answer    : Matrices;
      Row       : Row_Indices;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --handle first diagonal element
      Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
         Left(Row_Indices'FIRST, Col_Indices'FIRST) +
         Right(Row_Indices'FIRST, Col_Indices'FIRST);

--    --addition calculations will only be carried out on the bottom half
--    --of the input matrices followed by assignments to the symmetric
--    --elements in the top half of the matrix

--    --as Row goes down the rows, S_Col will go across the columns to keep
```

```
--    --track of the column containing the diagonal element
      S_Col := Col_Indices'SUCC(Col_Indices'FIRST);
      Row := Row_Indices'SUCC(Row_Indices'FIRST);
      Do_All_Rows_Except_First:
         loop

--          --as Col goes across the columns, S_Row will go down the rows;
            S_Row := Row_Indices'FIRST;
            Add_Bottom_Half_Elements:
               for Col in Col_Indices'FIRST ..
                          Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop

--                --add elements in bottom half of the matrix
                  Answer(Row,Col) := Left(Row, Col) + Right(Row, Col);

--                --assign value to symmetric element in top half of matrix
                  Answer(S_Row,S_Col) := Answer(Row, Col);

                  S_Row     := Row_Indices'SUCC(S_Row);

               end loop Add_Bottom_Half_Elements;

--          --add diagonal elements together
            Answer(Row, S_Col) := Left(Row,S_Col) + Right(Row,S_Col);

            exit when Row = Row_Indices'LAST;
            S_Col := Col_Indices'SUCC(S_Col);
            Row := Row_Indices'SUCC(Row);

         end loop Do_All_Rows_Except_First;

      return Answer;

   end "+";
```

3.3.6.2.9.19.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.6.8 LIMITATIONS

None.

3.3.6.2.9.19.10.7 "-" UNIT DESIGN (CATALOG #P407-0)

This function subtracts two symmetric input matrices by subtracting the individual elements of the input matrices, taking advantage of their symmetricity.

3.3.6.2.9.19.10.7.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Matrix to be subtracted from
Right	Matrices	In	Matrix to be used as the subtrahend

3.3.6.2.9.19.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding two input matrices
Row	Row_Indices	N/A	Index into first dimension of matrix
S_Col	Col_Indices	N/A	Used to keep track of column containing diagonal element for the current row
S_Row	Row_Indices	N/A	When used with S_Col, marks the symmetric counterpart to the element being referenced in the bottom half of the array; i.e., A(row,col) = A(s_row,s_col)

3.3.6.2.9.19.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    --------------------
--    --declaration section
--    --------------------

      Answer    : Matrices;
      Row       : Row_Indices;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

--    --handle first diagonal element
      Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
         Left(Row_Indices'FIRST, Col_Indices'FIRST) -
         Right(Row_Indices'FIRST, Col_Indices'FIRST);

--    --subtraction calculations will only be carried out on the bottom half
--    --of the input matrices followed by assignments to the symmetric
--    --elements in the top half of the matrix

--    --as Row goes down the rows, S_Col will go across the columns to keep
--    --track of the column containing the diagonal element
      S_Col := Col_Indices'SUCC(Col_Indices'FIRST);
      Row := Row_Indices'SUCC(Row_Indices'FIRST);
      Do_All_Rows_Except_First:
         loop
```

```
--          --as Col goes across the columns, S_Row will go down the rows;
            S_Row := Row_Indices'FIRST;
            Subtract_Bottom_Half_Elements:
               for Col in Col_Indices'FIRST ..
                          Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop

--             --subtract elements in bottom half of the matrix
               Answer(Row,Col) := Left(Row, Col) - Right(Row, Col);

--             --assign value to symmetric element in top half of matrix
               Answer(S_Row,S_Col) := Answer(Row, Col);

               S_Row     := Row_Indices'SUCC(S_Row);

            end loop Subtract_Bottom_Half_Elements;

--          --subtract diagonal elements together
            Answer(Row, S_Col) := Left(Row,S_Col) - Right(Row,S_Col);

            exit when Row = Row_Indices'LAST;
            S_Col := Col_Indices'SUCC(S_Col);
            Row := Row_Indices'SUCC(Row);

         end loop Do_All_Rows_Except_First;

      return Answer;

   end "-";
```

3.3.6.2.9.19.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.7.8 LIMITATIONS

None.

3.3.6.2.9.20 VECTOR_SCALAR_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P422-0)

This package provides a set of functions to multiply and divide each element of a vector by a scalar.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.20.1 REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the units in this part:

Name	Requirements Allocation
"*"	R065
"/"	R066

3.3.6.2.9.20.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.20.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types previously in this part's package specification:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices	discrete type	Used to dimension vectors
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars
"/"	function	Used to define the operation Elements2 := Elements1 / Scalars

3.3.6.2.9.20.4 LOCAL DATA

None.

3.3.6.2.9.20.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.20.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Scalar_Operations_Constrained is

end Vector_Scalar_Operations_Constrained;
```

3.3.6.2.9.20.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.20.8 LIMITATIONS

None.

3.3.6.2.9.20.9 LLCSC DESIGN

None.

3.3.6.2.9.20.10 UNIT DESIGN

3.3.6.2.9.20.10.1 "*" UNIT DESIGN (CATALOG #P423-0)

This function calculates a scaled vector by multiplying each element of an input vector by a scale factor.

3.3.6.2.9.20.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R065.

3.3.6.2.9.20.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.20.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Vectors2	In	Vector to be scaled
Multiplier	Scalars	In	Scale factor

3.3.6.2.9.20.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors1	Scaled vector

3.3.6.2.9.20.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.20.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Vector     : Vectors2;
                 Multiplier : Scalars) return Vectors1 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Vectors1;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

     Process:
        for Index in Indices loop

           Answer(Index) := Vector(Index) * Multiplier;

        end loop Process;

     return Answer;

   end "*";
```

3.3.6.2.9.20.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Vector_Scalar_Operations_-Constrained:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices	discrete type	Used to dimension vectors
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subprograms to Vector_Scalar_Operations_Constrained package:

Name	Type	Description
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars

3.3.6.2.9.20.10.1.8 LIMITATIONS

None.

3.3.6.2.9.20.10.2 "/" UNIT DESIGN (CATALOG #P424-0)

This function calculates a scaled vector by dividing each element of an input vector by a scale factor.

3.3.6.2.9.20.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R066.

3.3.6.2.9.20.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.20.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Vectors1	In	Vector to be scaled
Divisor	Scalars	In	Scale factor

3.3.6.2.9.20.10.2.4 LOCAL DATA

Data objects:

The following describes the local data maintained by this part:

Name	Type	Description
Answer	Vectors2	Scaled vector

3.3.6.2.9.20.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.20.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "/" (Vector  : Vectors1;
                 Divisor : Scalars) return Vectors2 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Vectors2;

-- ------------------------------------
-- --begin function Vector_Scalar_Divide
-- ------------------------------------

   begin

      Process:
         for Index in Indices loop

            Answer(Index) := Vector(Index) / Divisor;

         end loop Process;

      return Answer;

   end "/";
```

3.3.6.2.9.20.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Vector_Scalar_Operations_-Constrained:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices	discrete type	Used to dimension vectors
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subprograms to Vector_Scalar_Operations_Constrained package:

Name	Type	Description
"/"	function	Used to define the operation Elements2 := Elements1 / Scalars

3.3.6.2.9.20.10.2.8 LIMITATIONS

None.

3.3.6.2.9.21 MATRIX_SCALAR_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P428-0)

This package provides a set of functions which will scale a matrix by multiplying or dividing each element of the matrix by a scale factor.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.21.1 REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the parts in this LLCSC:

Name	Requirements Allocation
"*"	R073
"/"	R074

3.3.6.2.9.21.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.21.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described in this part's package specification:

Data types:

Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col_Indices	discrete type	Used to dimension second dimension of matrices
Row_Indices	discrete type	Used to dimension first dimension of matrices
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms:

Name	Type	Description
"*"	function	Function to define the operation Elements1 * Scalars := Elements2
"/"	function	Function to define the operation Elements2 / Scalars := Elements1

3.3.6.2.9.21.4 LOCAL DATA

None.

3.3.6.2.9.21.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.21.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Scalar_Operations_Constrained is

end Matrix_Scalar_Operations_Constrained;
```

3.3.6.2.9.21.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

3.3.6.2.9.21.8 LIMITATIONS

None.

3.3.6.2.9.21.9 LLCSC DESIGN

None.

3.3.6.2.9.21.10 UNIT DESIGN

3.3.6.2.9.21.10.1 "*" UNIT DESIGN (CATALOG #P429-0)

This function calculates a scaled matrix by multiplying each element of an input matrix by a scalar.

3.3.6.2.9.21.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R073.

3.3.6.2.9.21.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.21.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices1	In	Matrix to be scaled
Multiplier	Scalars	In	Scale factor

3.3.6.2.9.21.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices2	Scaled matrix

3.3.6.2.9.21.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.21.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Matrix     : Matrices1;
                 Multiplier : Scalars) return Matrices2 is

--   ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices2;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for Col in Col_Indices loop
```

```
                Answer(Row, Col) := Matrix(Row, Col) * Multiplier;

            end loop Col_Loop;

        end loop Row_Loop;

    return Answer;

end "*";
```

3.3.6.2.9.21.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Matrix_Scalar_Operations:

Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col_Indices	discrete type	Used to dimension second dimension of matrices
Row_Indices	discrete type	Used to dimension first dimension of matrices
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Matrix_Scalar_Operations_Constrained package.

Name	Type	Description
"*"	function	Function to define the operation Elements1 * Scalars := Elements2

3.3.6.2.9.21.10.1.8 LIMITATIONS

None.

3.3.6.2.9.21.10.2 "/" UNIT DESIGN (CATALOG #P430-0)

This function calculates a scaled matrix by dividing each element of an input matrix by a scale factor.

3.3.6.2.9.21.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R074.

3.3.6.2.9.21.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.21.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices2	In	Matrix to be scaled
Divisor	Scalars	In	Scale factor

3.3.6.2.9.21.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices1	Scaled matrix

3.3.6.2.9.21.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.21.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "/" (Matrix  : Matrices2;
                 Divisor : Scalars) return Matrices1 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices1;

-- --------------------
-- --begin function "/"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for Col in Col_Indices loop

                  Answer(Row, Col) := Matrix(Row, Col) / Divisor;

               end loop Col_Loop;

         end loop Row_Loop;

      return Answer;

   end "/";
```

3.3.6.2.9.21.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Matrix_Scalar_Operations:

Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col_Indices	discrete type	Used to dimension second dimension of matrices
Row_Indices	discrete type	Used to dimension first dimension of matrices
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Matrix_Scalar_Operations_Constrained package.

Name	Type	Description
"*"	function	Function to define the operation Elements1 * Scalars := Elements2
"/"	function	Function to define the operation Elements2 / Scalars := Elements1

3.3.6.2.9.21.10.2.8 LIMITATIONS

None.

3.3.6.2.9.22 VECTOR_MATRIX_MULTIPLY_UNRESTRICTED PACKAGE DESIGN (CATALOG #P437-0)

This package contains a function which multiplies a 1 x m vector by an m x n matrix producing a 1 x n vector. If the length of the vector is not the same as the length of the first dimension of the matrix a DIMENSION_ERROR exception is raised. None of the ranges need to be the same.

The function in this package can be made to handle sparse matrices and/or vectors by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
"*"	P1051-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.22.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.22.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains code which checks the lengths of the indices used to instantiate the package to ensure the sizes of the input vector, input matrix, and output vector are compatible with the operation to be performed.

3.3.6.2.9.22.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of General_Vector_Matrix_Algebra:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Input_Vector_Elements	floating point type	Type of elements in the input vector
Matrix_Elements	floating point type	Type of elements in the input matrix
Output_Vector_Elements	floating point type	Type of elements in the output vector
Col_Indices	discrete type	Used to dimension second dimension of input matrix
Row_Indices	discrete type	Used to dimension first dimension of input matrix
Input_Vector_Indices	discrete	Used to dimension input vector
Output_Vector_Indices	discrete	Used to dimension output vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Input_V Elements * Matrix_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

FORMAL PARAMETERS:

The following table describes the formal parameters for the "*" unit contained in this part:

Name	Type	Description
Vector	Input_Vectors	1xm vector to be used in the calculation
Matrix	Input_Matrices	mxn matrix to be usedin the calculation

3.3.6.2.9.22.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by the unit in this part:

Name	Type	Value	Description
Answer	Output_Vectors	N/A	Result vector being calculated
M_V	Input_Vector_ Indices	N/A	Index into the 1 x m input vector
N_A	Output_Vector_ Indices	N/A	Index into the n x 1 output vector
N	Col_Indices	N/A	Column index into the m x n input matrix
M	Row_Indices	N/A	Row index into the m x n input matrix

3.3.6.2.9.22.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.22.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Matrix_Multiply_Unrestricted is

   function "*" (Vector : Input_Vectors;
                 Matrix : Input_Matrices) return Output_Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Output_Vectors := (others => 0.0);
      M_V    : Input_Vector_Indices;
      N_A    : Output_Vector_Indices;
      N      : Col_Indices;
      M      : Row_Indices;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

      N_A := Output_Vector_Indices'FIRST;
      N   := Col_Indices'FIRST;
      N_Loop:
         loop

            M_V := Input_Vector_Indices'FIRST;
            M   := Row_Indices'FIRST;
            M_Loop:
               loop

                  Answer (N_A) := Answer(N_A) + Vector(M_V) * Matrix(M, N);

                  exit when M = Row_Indices'LAST;
                  M := Row_Indices'SUCC(M);
                  M_V := Input_Vector_Indices'SUCC(M_V);

               end loop M_Loop;

            exit when N = Col_Indices'LAST;
            N := Col_Indices'SUCC(N);
            N_A := Output_Vector_Indices'SUCC(N_A);

         end loop N_Loop;

      return Answer;
```

```
   end "*";

-------------------------------------------------
--begin package Vector_Matrix_Multiply_Unrestricted
-------------------------------------------------

begin

-- --make sure package has been instantiated with the correct dimensions;
-- --the following dimensions are expected:  [1xm] * [mxn] => [1xn]

   if Input_Vectors'LENGTH /= Input_Matrices'LENGTH(1) or      --m's not equal
      Input_Matrices'LENGTH(2) /= Output_Vectors'LENGTH then   --n's not equal

      raise Dimension_Error;

   end if;

end Vector_Matrix_Multiply_Unrestricted;
```

3.3.6.2.9.22.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description
Dimension_Error	Raised by a routine or package when input received has dimensions incompatible with the type of operation to be performed

3.3.6.2.9.22.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the sizes of the data objects are incompatible for the multiplication operation

3.3.6.2.9.22.9 LLCSC DESIGN

None.

3.3.6.2.9.22.10 UNIT DESIGN

None.

3.3.6.2.9.23 ABA_TRANS_DYNAM_SPARSE_MATRIX_SQ_MATRIX PACKAGE DESIGN (CATALOG #P1066-0)

This package contains a function which does an ABA transpose multiply on a dynamically sparse matrix (m x n) and a square (n x n) matrix, yielding a square matrix. The first multiply (A*B) is constrained and the second (AB * transpose A) is restricted.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
ABA_Transpose	P1067-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.23.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.23.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.23.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
A_Elements	floating point type	Type of element in the dymamically sparse input matrix
B_Elements	floating point type	Type of element in the square input matrix
C_Elements	floating point type	Type of element in the output vector
M_Indices	discrete type	Used to dimension the 1st dimension of the sparse input matrix
N_Indices	discrete type	Used to dimension the 2nd dimension of the sparse matrix and both dimensions of the square matrix
A_Matrices	array	Data type of the dynamically sparse input matrix
B_Matrices	array	Data type of the square input matrix
C_Matrices	array	Data type of the output matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function defining the operation A_Elements * B_Elements := C_Elements
"*"	function	Function defining the operation C_Elements * A_Elements := C_Elements

3.3.6.2.9.23.4 LOCAL DATA

None.

3.3.6.2.9.23.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.23.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix is


   function Sparse_Left_Multiply(Left  : A_Elements;
                                 Right : B_Elements ) return A_Elements is
      Answer : A_Elements;
```

```
   begin
      If Left = 0.0 then
         Answer := 0.0;
      else
         Answer := Left * A_Elements( Right );
      end if;
      return Answer;
   end Sparse_Left_Multiply;



   function Sparse_Right_Multiply( Left  : A_Elements;
                                   Right : A_Elements ) return C_Elements is
      Answer : C_Elements;
   begin
      If Right = 0.0 then
         Answer := 0.0;
      else
         Answer := C_Elements( Left * Right );
      end if;
      return Answer;
   end Sparse_Right_Multiply;



   function Matrix_Multiply is new Matrix_Matrix_Multiply_Restricted
      ( Left_Elements   => A_Elements,
        Right_Elements  => B_Elements,
        Output_Elements => A_Elements,
        M_Indices       => M_Indices,
        N_Indices       => N_Indices,
        P_Indices       => N_Indices,
        Left_Matrices   => A_Matrices,
        Right_Matrices  => B_Matrices,
        Output_Matrices => A_Matrices,
        "*"             => Sparse_Left_Multiply );



   function Matrix_Transpose_Multiply is new
      Matrix_Matrix_Transpose_Multiply_Restricted
         ( Left_Elements   => A_Elements,
           Right_Elements  => A_Elements,
           Output_Elements => C_Elements,
           M_Indices       => M_Indices,
           N_Indices       => N_Indices,
           P_Indices       => M_Indices,
           Left_Matrices   => A_Matrices,
           Right_Matrices  => A_Matrices,
           Output_Matrices => C_Matrices,
           "*"             => Sparse_Right_Multiply );


   function ABA_Transpose( A : A_Matrices;
                           B : B_Matrices )
                                return C_Matrices is
```

```
      Intermediate : A_Matrices;
      Answer       : C_Matrices;

   begin

--    ------------------
--    - multiply A * B -
--    ------------------
      Intermediate := Matrix_Multiply( Left  => A,
                                       Right => B );


--    --------------------------------
--    - multiply AB * transpose of A -
--    --------------------------------
      Answer := Matrix_Transpose_Multiply( Left  => Intermediate,
                                           Right => A );
      return Answer;

   end ABA_Transpose;

end ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix;
```

3.3.6.2.9.23.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in this top-level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in ancestral units:

Name	Type	Source	Description
Matrix_Matrix_Multiply_ Restricted	generic function	GVMA	Used to multiply the sparse matrix by the square matrix
Matrix_Matrix_Transpose_ Multiply	generic function	GVMA	Used to multiply the product of the first operation by the transpose of the sparse matrix

3.3.6.2.9.23.8 LIMITATIONS

None.

3.3.6.2.9.23.9 LLCSC DESIGN

None.

3.3.6.2.9.23.10 UNIT DESIGN

None.

3.3.6.2.9.24 ABA_TRANS_VECTOR_SQ_MATRIX PACKAGE DESIGN (CATALOG #P1068-0)

This package contains a function which does an ABA transpose multiply on a vector (1 x m) and a square (m x m) matrix, yielding a scalar value.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
ABA_Transpose	P1069-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.24.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.24.2 LOCAL ENTITIES DESIGN

Subprograms:

The following table summarizes the subroutines which are local to this part:

Name	Type	Description
Multiply_VM	function	Function defining the operation Vector_ Elements * Matrix_Elements := Vector_ Elements
Multiply_VV	function	Function defining the operation Vector_ Elements * Vector_Elements := Scalars
Vector_Matrix_ Multiply	function	Function defining a vector matrix multiplication Input_Vectors * Input_Matrices := Output_Vectors. Instantiation of GVMA. Vector_Matrix_Multiply_Restricted
Vector_Vector_ Multiply	function	Function defining a vector vector multiply (dot product) Vectors * Vectors := Scalars Instantiation of GVMA.Dot_Product_ Operations_Restricted

3.3.6.2.9.24.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Vector_Elements	floating point type	Type of element in the input vector.
Matrix_Elements	floating point type	Type of element in the sqaure input matrix.
Scalars	floating point type	Type of element in the output scalar
Indices	discrete type	Used to dimension the input vector and both dimensions of the input matrix
Vectors	array	Data type of the input vector
Matrices	array	Data type of the square input matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function defining the operation Vector_Elements * Matrix_Elements := Vector_Elements
"*"	function	Function defining the operation Vector_Elements * Vector_Elements := Scalars

3.3.6.2.9.24.4 LOCAL DATA

None.

3.3.6.2.9.24.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.24.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body ABA_Trans_Vector_Sq_Matrix is


   function Multiply_VM( Left  : Vector_Elements;
```

```
                        Right : Matrix_Elements ) return Vector_Elements is
   begin
      return Left * Vector_Elements( Right );
   end Multiply_VM;



   function Multiply_VV( Left : Vector_Elements;
                         Right: Vector_Elements ) return Scalars is
   begin
      return Scalars( Left ) * Scalars( Right );
   end Multiply_VV;



   function Vector_Matrix_Multiply is new Vector_Matrix_Multiply_Restricted
           ( Input_Vector_Elements  => Vector_Elements,
             Matrix_Elements        => Matrix_Elements,
             Output_Vector_Elements => Vector_Elements,
             Indices1               => Indices,
             Indices2               => Indices,
             Input_Vectors          => Vectors,
             Input_Matrices         => Matrices,
             Output_Vectors         => Vectors,
             "*"                    => Multiply_VM );



   function Vector_Vector_Multiply is new Dot_Product_Operations_Restricted
              ( Left_Elements   => Vector_Elements,
                Right_Elements  => Vector_Elements,
                Result_Elements => Scalars,
                Indices         => Indices,
                Left_Vectors    => Vectors,
                Right_Vectors   => Vectors,
                "*"             => Multiply_VV );


   function ABA_Transpose( A : Vectors;
                           B : Matrices ) return Scalars is

      Partial_Answer : Vectors;
      Answer         : Scalars;

   begin

--    ------------------
--    - multiply A * B -
--    ------------------
      Partial_Answer := Vector_Matrix_Multiply( Vector => A,
                                                Matrix => B );


--    -------------------------------
--    - multiply AB * transpose of A -
--    -------------------------------
      Answer := Vector_Vector_Multiply( Left  => Partial_Answer,
```

```
                                        Right => A );

      return Answer;

   end ABA_Transpose;

end ABA_Trans_Vector_Sq_Matrix;
```

3.3.6.2.9.24.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in this top-level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in ancestral units:

Name	Type	Source	Description
Vector_Matrix_Multiply_ Restricted	generic function	GVMA	Used to multiply the input vector by the square matrix
Dot_Product_Multiply_ Restriced	generic function	GVMA	Used to multiply the product of the first operation by the transpose of the input vector

3.3.6.2.9.24.8 LIMITATIONS

None.

3.3.6.2.9.24.9 LLCSC DESIGN

None.

3.3.6.2.9.24.10 UNIT DESIG"

None.

3.3.6.2.9.25 ABA_TRANS_VECTOR_SCALAR PACKAGE DESIGN (CATALOG #P1070-0)

This package contains a function which does an ABA transpose multiply on a vector (m x 1) and a scalar value, yielding a square (m x m) matrix.

The following table lists the catalog numbers for subunits contained in this part:

Name	Catalog _#
ABA_Transpose	P1071-0

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.25.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.25.2 LOCAL ENTITIES DESIGN

Packages:

The following table summarizes the packages which are local to this part:

Name	Type	Description
VS_Opns	package	Package defining Vector Scalar operations. Only multiply operator is used. Instantiation of GVMA.Vector_Scalar_Operations_Constrained

Subprograms:

The following table summarizes the subroutines which are local to this part:

Name	Type	Description
Multiply_VS	function	Function defining the operation Vector_Elements * Scalars := Vector_Elements. Used in instantiation. Used in instantiation of Vector_Scalar_Operations_Constrained
Divide_VS	function	Function provided defining operation Vector_Elements / Scalars := Scalars. Provided for instantiation. Not used in any computations. Used in instantiation of Vector_Scalar_Operations_Constrained
VV_Transpose_Multiply	function	Function defining a matrix transpose multiply A_Matrices * transpose A_Matrices := C_Matrices. Instantiation of GVMA.Matrix_Matrix_Transpose_Multiply

3.3.6.2.9.25.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Vector_Elements	floating point type	Type of element in the input vector.
Matrix_Elements	floating point type	Type of element in the square output matrix.
Scalars	floating point type	Type of element in the output scalar
Indices	discrete type	Used to dimension the input vector and the square output matrix
Vectors	array	Data type of the input vector
Matrices	array	Data type of the square output matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function defining the operation Vector_Elements * Scalars := Vector_Elements
"*"	function	Function defining the operation Vector_Elements * Vector_Elements := Matrix_Elements

3.3.6.2.9.25.4 LOCAL DATA

None.

3.3.6.2.9.25.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.25.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body ABA_Trans_Vector_Scalar is

-- ----------------------------------------
-- -- Operators provided for instantiations -
```

```
-----------------------------------------

   function Multiply_VS( Left  : Vector_Elements;
                         Right : Scalars )  return Vector_Elements is
   begin
      return Left * Vector_Elements( Right );
   end Multiply_VS;


-- ------------------------------------------------------------------------
-- -- This operator is not used, but is required for the instantiation. -
-- -- It is "dummied" out to make it as small as possible.              -
-- ------------------------------------------------------------------------
   function Divide_VS( Left  : Vector_Elements;
                       Right : Scalars )  return Vector_Elements is
   begin
      return Left;
   end Divide_VS;


   function Multiply_VV( Left  : Vector_Elements;
                         Right : Vector_Elements ) return Matrix_Elements is
   begin
      return Matrix_Elements( Left ) * Matrix_Elements( Right );
   end Multiply_VV;


-- -------------------------------------
-- -- Instantiations for ABA transpose -
-------------------------------------

   package VS_Opns is new Vector_Scalar_Operations_Constrained
               ( Elements1 => Vector_Elements,
                 Element 2 => Vector_Elements,
                 Scalars   => Scalars,
                 Indices   => Indices,
                 Vectors1  => Vectors,
                 Vectors2  => Vectors
                 "*"       => Multipl _VS,
                 "/"       => Divide_VS );

   use VS_Opns;


   function VV_Transpose_Multiply is new
      Vector_Vector_Transpose_Multiply_Restricted
         ( Left_Vector_Elements  => Vector_Elements,
           Right_Vector_Elements => Vector_Elements,
           Matrix_Elements       => Matrix_Elements,
           Indices1              => Indices,
           Indices2              => Indices,
           Left_Vectors          => Vectors,
           Right_Vectors         => Vectors,
           Matrices              => Matrices,
           "*"                   => Multiply_VV );
```

```
   function ABA_Transpose( A : Vectors;
                           B : Scalars ) return Matrices is

      Partial_Answer : Vectors;
      Answer         : Matrices;

   begin

--    ------------------
--    - multiply A * B -
--    ------------------
      Partial_Answer := A * B;

--    --------------------------------
--    - multiply AB * transpose of A -
--    --------------------------------
      Answer := VV_Transpose_Multiply( Left  => Partial_Answer,
                                       Right => A );
      return Answer;

   end ABA_Transpose;

end ABA_Trans_Vector_Scalar;
```

3.3.6.2.9.25.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in this top-level component:

Packages:

The following table summarizes the external packages required by this part:

Name	Type	Source	Description
Vector_Scalar_ Operations_ Constrained	generic package	GVMA	Package allowing operation Vectors * Scalars := Vectors.

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

Name	Type	Source	Description
Matrix_Matrix_ Transpose_ Multiply	generic function	GVMA	Function allowing the operation vector * transpose vector := Matrices.

3.3.6.2.9.25.8 LIMITATIONS

None.

3.3.6.2.9.25.9 LLCSC DESIGN

None.

3.3.6.2.9.25.10 UNIT DESIGN

None.

3.3.6.2.9.26 COLUMN_MATRIX_OPERATIONS PACKAGE DESIGN (CATALOG #P1072-0)

This package defines a column matrix which contains a column vector which is set on one of the columns of the matrix and a diagonal, which can only have the values of 1 or 0 on the diagonal. It provides operations on that type. See the top level decomposition section for a list of the operations provided.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.26.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.26.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.26.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Vector_Elements	floating point type	Type of element in the column matrix's column vector
Indices	discrete type	Used to dimension the column matrix and the vector in the column matrix
Vectors	array	Data type of the vector in the column matrix

3.3.6.2.9.26.4 LOCAL DATA

None.

3.3.6.2.9.26.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.26.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Column_Matrix_Operations is

end Column_Matrix_Operations;
```

3.3.6.2.9.26.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.26.8 LIMITATIONS

None.

3.3.6.2.9.26.9 LLCSC DESIGN

None.

3.3.6.2.9.26.10 UNIT DESIGN

3.3.6.2.9.26.10.1 SET_DIAGONAL_AND_SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P1073-0)

This function provides the initialization of a column matrix with values of a vector (to be subtracted from the identity matrix) and the diagonal to be set to 1's.

3.3.6.2.9.26.10.1.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.26.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.26.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Mode	Type	Description
Column	in	Vectors	The vector to be set on the specified column
Active_Column	in	Indices	Value designating which column is to be set

3.3.6.2.9.26.10.1.4 LOCAL DATA

Data objects:

The following table describes the objects maintained by this part:

Name	Type	Description
Answer	Column_Matrices	The resultant column matrix

3.3.6.2.9.26.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.26.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Set_Diagonal_and_Subtract_from_Identity
   ( Column        : Vectors;
     Active_Column : Indices ) return Column_Matrices is

   Answer : Column_Matrices;

begin

   Answer.Col_Vector    := Column;
   Answer.Diagonal      := TRUE;
   Answer.Active_Column := Active_Column;
   Range_Loop:
      For Index in Indices loop
         Answer.Col_Vector( Index ) := - Answer.Col_Vector( Index );
      end loop Range_Loop;
   Answer.Col_Vector(Active_Column) := Answer.Col_Vector(Active_Column) +
                                       1.0;
   return Answer;

end Set_Diagonal_and_Subtract_from_Identity;
```

3.3.6.2.9.26.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.26.10.1.8 LIMITATIONS

None.

3.3.6.2.9.26.10.2 ABA_TRANSPOSE UNIT DESIGN (CATALOG #P1075-0)

This package contains a function which does an ABA transpose multiply on a column matrix (m x m) and a square (m x m) matrix, yielding a square matrix (m x m). The matrix multiplies (A*B) and (AB * transpose A) are restricted.

3.3.6.2.9.26.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R.

3.3.6.2.9.26.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.26.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
B_Matrix_Elements	floating point type	Type of element in the square input matrix
C_Matrix_Elements	floating point type	Type of element in the output vector
B_Matrices	array	Data type of the square input matrix
C_Matrices	array	Data type of the output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function defining the operation Vector_Elements * B_Matrix_Elements := B_Matrix_Elements

FORMAL PARAMETERS:

The following table describes the formal parameters for this part:

Name	Type	Description
A	Column_Matrices	The input column matrix
B	B_Matrices	The input square matrix

3.3.6.2.9.26.10.2.4 LOCAL DATA

None.

3.3.6.2.9.26.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.26.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function ABA_Transpose( A : Column_Matrices;
                        B : B_Matrices ) return C_Matrices is

   Answer      : C_Matrices;
   Temp_Vector : Vectors := A.Col_Vector;

begin
   if A.Diagonal then
      Temp_Vector( A.Active_Column ) := Temp_Vector( A.Active_Column ) - 1.0;
      M_Loop:
         For Row in Indices loop
            P_Loop:
               For Col in Indices loop
                  Answer( Row, Col ) := C_Matrix_Elements(
                     Temp_Vector( Row ) * Temp_Vector( Col ) *
                     B( A.Active_Column, A.Active_Column )          +
                     Temp_Vector( Col ) * B( A.Active_Column, Row ) +
                     Temp_Vector( Row ) * B( A.Active_Column, Col ) +
                     B( Row, Col ) );
               end loop P_Loop;
         end loop M_Loop;
   else
```

```
         M1_Loop:
            For Row in Indices loop
               P1_Loop:
                  For Col in Indices loop
                     Answer( Row, Col ) := C_Matrix_Elements(
                        A.Col_Vector( Row ) * A.Col_Vector( Col ) *
                        B( A.Active_Column, A.Active_Column ) );
                  end loop P1_Loop;
            end loop M1_Loop;
      end if;
      return Answer;

   end ABA_Transpose;
```

3.3.6.2.9.26.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.26.10.2.8 LIMITATIONS

None.

3.3.6.2.9.26.10.3 ABA_SYMM_TRANSPOSE UNIT DESIGN (CATALOG #P1076-0)

This package contains a function which does an ABA transpose multiply on a column matrix (m x m) and a symmetric (m x m) matrix, yielding a symmetric matrix (m x m). The matrix multiplies (A*B) and (AB * transpose A) are restricted.

3.3.6.2.9.26.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R.

3.3.6.2.9.26.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.26.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
B_Matrix_Elements	floating point type	Type of element in the square input matrix
C_Matrix_Elements	floating point type	Type of element in the output vector
B_Matrices	array	Data type of the square input matrix
C_Matrices	array	Data type of the output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function defining the operation Vector_Elements * B_Matrix_Elements := B_Matrix_Elements

FORMAL PARAMETERS:

The following table describes the formal parameters for this part:

Name	Type	Description
A	Column_Matrices	The input column matrix
B	B_Matrices	The input symmetric matrix

3.3.6.2.9.26.10.3.4 LOCAL DATA

None.

3.3.6.2.9.26.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.26.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
function ABA_Symm_Transpose( A : Column_Matrices;
                             B : B_Matrices ) return C_Matrices is

   Answer      : C_Matrices;
```

```
      Last        : Indices;
      Temp_Vector : Vectors := A.Col_Vector;

   begin

      Last := Indices'LAST;
      if A.Diagonal then     -- Diagonal value is 1 --
         Temp_Vector( A.Active_Column ) := Temp_Vector( A.Active_Column ) - 1.0;
         M_Loop:
            For Row in Indices loop
               P_Loop:
--                ---------------------
--                - Calculate values -
--                ---------------------
                  For Col in Row .. Indices'LAST loop
                     Answer( Row, Col ) := C_Matrix_Elements(
                        Temp_Vector( Row ) * Temp_Vector( Col ) *
                        B( A.Active_Column, A.Active_Column )          +
                        Temp_Vector( Col ) * B( A.Active_Column, Row ) +
                        Temp_Vector( Row ) * B( A.Active_Column, Col ) +
                        B( Row, Col ) );
--                   ------------------------------------------
--                   - Assign calculated value to corresponding -
--                   - lower triangular position                -
--                   ------------------------------------------
                     Answer( Col, Row ) := Answer( Row, Col );
                  end loop P_Loop;
            end loop M_Loop;
      else                        -- diagonal value is 0 --
         M1_Loop:
            For Row in Indices loop
               P1_Loop:
--                ---------------------
--                - Calculate values -
--                ---------------------
                  For Col in Row .. Indices'LAST loop
                     Answer( Row, Col ) := C_Matrix_Elements(
                        A.Col_Vector( Row ) * A.Col_Vector( Col ) *
                        B( A.Active_Column, A.Active_Column ) );
--                   ------------------------------------------
--                   - Assign calculated value to corresponding -
--                   - lower triangular position                -
--                   ------------------------------------------
                     Answer( Col, Row ) := Answer( Row, Col );
                  end loop P1_Loop;
            end loop M1_Loop;
      end if;                     -- Diagonal value is 0 --
      return Answer;

   end ABA_Symm_Transpose;
```

3.3.6.2.9.26.10.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.26.10.3.8 LIMITATIONS

None.

3.3.6.2.10 UNIT DESIGN

3.3.6.2.10.1 MATRIX_MATRIX_MULTIPLY_RESTRICTED UNIT DESIGN (CATALOG #P441-0)

This function multiplies an m x n matrix by an n x p matrix, returning an m x p matrix.

The result of this operation is defined as follows:

```
a(i,j) := b(i,k) * c(k,j)
```

3.3.6.2.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.3.6.2.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification for General_Vector_Matrix_Algebra.

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Data type of elements in left input matrix
Right_Elements	floating point type	Data type of elements in right input matrix
Output_Elements	floating point type	Data type of elements in output matrix
M_Indices	discrete type	Used to dimension first dimension of left input matrix and output matrix
N_Indices	discrete type	Used to dimension second dimension of left input matrix and first dimension of right input matrix
P_Indices	discrete type	Used to dimension second dimension of right input matrix and output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part. To tailor this function to handle sparse matrices, the formal subroutines should be set up to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Left_Elements * Right_Elements := Output_Elements
"+"	function	Function defining the operation Output_Elements + Output_Elements := Output_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Matrices	In	m x n matrix
Right	Right_Matrices	In	n x p matrix

3.3.6.2.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Output_Matrices	N/A	Result matrix

3.3.6.2.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
function Matrix_Matrix_Multiply_Restricted (Left  : Left_Matrices;
                Right : Right_Matrices) return Output_Matrices is

-- ----------------------
-- --declaration section-
-- ----------------------

   Answer      : Output_Matrices;

-------------------------------------------------
--begin of function Matrix_Matrix_Multiply_Restricted
-------------------------------------------------

begin

   Answer := (others => (others => 0.0));

   M_Loop:
      for M in M_Indices loop

         P_Loop:
            for P in P_Indices loop

               N_Loop:
                  for N in N_Indices loop

                     Answer(M, P) := Answer(M, P) +
                                     Left(M, N) * Right(N, P);

                  end loop N_Loop;

            end loop P_Loop;

      end loop M_Loop;

   return Answer;

end Matrix_Matrix_Multiply_Restricted;
```

3.3.6.2.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.1.8 LIMITATIONS

None.

3.3.6.2.10.2 MATRIX_VECTOR_MULTIPLY_RESTRICTED UNIT DESIGN (CATALOG #P436-0)

This function multiplies an m x n matrix by an n x 1 vector producing an m x 1 vector.

The result of this operation is defined as follows:

```
a(i) := b(i,j) * c(j)
```

3.3.6.2.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R069.

3.3.6.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following table describes this part's generic parameters which were previously described in the package specification of General_Vector_Matrix_-Algebra:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Indices1	discrete type	Used to dimension first dimension of input matrix and to dimension the output vector
Indices2	discrete type	Used to dimension second dimension of input matrix and to dimension the input vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Matrix_Elements * Input_Vector_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Input_Matrices	In	Matrix to be used as the multiplicand
Vector	Input_Vectors	In	Vector to be used as the multiplier

3.3.6.2.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Output_Vectors	Result of performing the matrix-vector multiplication

3.3.6.2.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
function Matrix_Vector_Multiply_Restricted
             (Matrix : Input_Matrices;
              Vector : Input_Vectors) return Output_Vectors is

-- ----------------------
-- --declaration section-
-- ----------------------

   Answer    : Output_Vectors;

-------------------------------------------------
--begin function Matrix_Vector_Multiply_Restricted
-------------------------------------------------

begin

   Answer := (others => 0.0);

   M_Loop:
      for M in Indices1 loop

         N_Loop:
            for N in Indices2 loop

               Answer(M) := Answer(M) +
                            Matrix(M, N) * Vector(N);

            end loop N_Loop;

      end loop M_Loop;

   return Answer;

end Matrix_Vector_Multiply_Restricted;
```

3.3.6.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.2.8 LIMITATIONS

None.

3.3.6.2.10.3 VECTOR_VECTOR_TRANSPOSE_MULTIPLY_RESTRICTED UNIT DESIGN (CATALOG #P444-0)

This function multiplies an m x 1 input vector by the transpose of a n x 1 input vector, returning the resultant m x n matrix.

The following defines the result of this operation:

```
a(i,j) := b(i) * c(j)
```

3.3.6.2.10.3.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification for General_Vector_Matrix_Algebra:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Vector_Elements	floating point type	Data type of elements in left input vector
Right_Vector_Elements	floating point type	Data type of elements in right input vector
Matrix_Elements	floating point type	Data type of elements in output matrix
Indices1	discrete type	Used to dimension left input vector and first dimension of output matrix
Indices2	discrete type	Used to dimension right input vector and second dimension of output matrix
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector
Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the multiplication operation Left_Vector_Elements * Right_Vector_Elements := Matrix_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Vectors	In	m x 1 vector
Right	Right_Vectors	In	1 x n vector

3.3.6.2.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result matrix

3.3.6.2.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
function Vector_Vector_Transpose_Multiply_Restricted
            (Left  : Left_Vectors ;
             Right : Right_Vectors) return Matrices is

-- ---------------------
-- --declaration section
-- ---------------------

   Answer   : Matrices;

------------------------------------------------------------
--begin function Vector_Vector_Transpose_Multiply_Restricted
------------------------------------------------------------

begin

   M_Loop:
      for M in Indices1 loop

         N_Loop:
            for N in Indices2 loop

               Answer(M, N) := Left(M) * Right(N);

            end loop N_Loop;

   end loop M_Loop;

   return Answer;

end Vector_Vector_Transpose_Multiply_Restricted;
```

3.3.6.2.10.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.3.8 LIMITATIONS

None.

3.3.6.2.10.4 MATRIX_MATRIX_TRANSPOSE_MULTIPLY_RESTRICTED UNIT DESIGN (CATALOG #P447-0)

This function multiples an m x n matrix by the transpose of a p x n matrix, returning the resultant m x p matrix.

The results of this operation are defined as follows:

```
a (i,j) := b(i,k) * c(j,k)
```

3.3.6.2.10.4.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described in the package specification for General_Vector_Matrix_Algebra:

Data types:

The following table describes the generic formal types required by this part:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input matrix
Right_Elements	floating point type	Type of elements in right input matrix
Output_Elements	floating point type	Type of elements in output matrix
M_Indices	discrete type	Used to dimension first dimension of left input matrix and output matrix
N_Indices	discrete type	Used to dimension second dimension of left and right input matrix
P_Indices	discrete type	Used to dimension first dimension of right input matrix and second dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator used to define the operation: Left_Elements * Right_Elements := Output_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Matrices	In	Matrix to be used as the multiplicand
Right	Right_Matrices	In	Matrix whose transpose is to be used as the multiplier

3.3.6.2.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Output_Matrices	N/A	Result matrix being calculated

3.3.6.2.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
function Matrix_Matrix_Transpose_Multiply_Restricted
           (Left  : Left_Matrices;
           Right : Right_Matrices) return Output_Matrices is

-- ---------------------
-- --declaration section
-- ---------------------

   Answer   : Output_Matrices;
```

```
------------------------------------------------------------
--begin function Matrix_Matrix_Transpose_Multiply_Restricted
------------------------------------------------------------

begin

   Answer := (others => (others => 0.0));

   M_Loop:
      for M in M_Indices loop

         P_Loop:
            for P in P_Indices loop

               N_Loop:
                  for N in N_Indices loop

                     Answer(M, P) := Answer(M, P) +
                                     Left(M, N) * Right(P, N);

                  end loop N_Loop;

            end loop P_Loop;

      end loop M_Loop;

   return Answer;

end Matrix_Matrix_Transpose_Multiply_Restricted;
```

3.3.6.2.10.4.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.4.8 LIMITATIONS

None.

3.3.6.2.10.5 DOT_PRODUCT_OPERATIONS_RESTRICTED UNIT DESIGN (CATALOG #P450-0)

This function performs a dot product operation on two m-element vectors.

3.3.6.2.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R063.

3.3.6.2.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification for General_Vector_Matrix_Algebra.

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input vector
Right_Elements	floating point type	Type of elements in right input vector
Result_Elements	floating point type	Data type of result of dot product
Indices	discrete type	Used to dimension input vectors
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Left_Elements * Right_Elements := Result_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Vectors	In	First vector in a dot product operation
Right	Right_Vectors	In	Second vector in a dot product operation

3.3.6.2.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Result_Elements	N/A	Result of dot product operation

3.3.6.2.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
function Dot_Product_Operations_Restricted
            (Left  : Left_Vectors;
             Right : Right_Vectors) return Result_Elements is

-- ----------------------
-- --declaration section-
-- ----------------------

   Answer  : Result_Elements;

-------------------------------------------------
--begin function Dot_Product_Operations_Restricted
-------------------------------------------------

begin

   Answer  := 0.0;

   Process:
      for Index in Indices loop

         Answer := Answer + Left(Index) * Right(Index);

      end loop Process;

   return Answer;

end Dot_Product_Operations_Restricted;
```

3.3.6.2.10.5.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.5.8 LIMITATIONS

None.

3.3.6.2.10.6 DIAGONAL_FULL_MATRIX_ADD_RESTRICTED UNIT DESIGN (CATALOG #P453-0)

This function adds an m-element diagonal matrix to an m x m matrix, returning the resultant m x m matrix.

3.3.6.2.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following table describes this part's generic parameters previously defined in the package specification for General_Vector_Matrix_Algebra:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Type of elements in input and output arrays
Diagonal_Range	integer type	Used to dimension Diagonal_Matrices
Indices	discrete type	Used to dimension input and output matrices
Diagonal_Matrices	array	Data type of diagonal input matrix
Full_Matrices	array	Data type of full input and output matrices

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
D_Matrix	Diagonal_Matrices	In	Input diagonal matrix
F_Matrix	Full_Input_Matrices	In	Input full matrix to be added to the diagonal matrix

3.3.6.2.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Full_Output_Matrices	N/A	Resultant matrix
Diag_Index	Diagonal_Range	N/A	Index into diagonal matrix
Index	Indices	N/A	Index into full matrix

3.3.6.2.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
function Diagonal_Full_Matrix_Add_Restricted
         (D_Matrix : Diagonal_Matrices;
          F_Matrix : Full_Matrices) return Full_Matrices is

-- ----------------------
-- --declaration section-
-- ----------------------

   Answer       : Full_Matrices;
   Diag_Index   : Diagonal_Range;
   Index        : Indices;

-------------------------------------------------
--begin function Diagonal_Full_Matrix_Add_Restricted
-------------------------------------------------

begin

-- --assign all values to answer and then add in diagonal elements
   Answer := F_Matrix;

-- --now add in diagonal elements

   Diag_Index := Diagonal_Range'FIRST;
   Index      := Indices'FIRST;
   Add_Loop:
      loop

         Answer(Index, Index) := Answer(Index, Index) + D_Matrix(Diag_Index);

         exit when Index = Indices'LAST;
```

```
        Diag_Index := Diagonal_Range'SUCC(Diag_Index);
        Index      := Indices'SUCC(Index);

      end loop Add_Loop;

   return Answer;

end Diagonal_Full_Matrix_Add_Restricted;
```

3.3.6.2.10.6.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.6.8 LIMITATIONS

None.

3.3.6.2.10.7 VECTOR_MATRIX_MULTIPLY_RESTRICTED UNIT DESIGN (CATALOG #P438-0)

This package contains a function which multiplies a 1 x m vector by an m x n matrix producing a 1 x n vector.

The calculations performed are as follows:

```
    c(j) := a(i) * b(i,j)
```

The function can be made to handle sparse matrices and/or vectors by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

3.3.6.2.10.7.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.7.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of General_Vector_Matrix_Algebra.

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Indices1	discrete type	Used to dimension first dimension of input matrix and to dimension the output vector
Indices2	discrete type	Used to dimension second dimension of input matrix and to dimension the input vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Input_Vector_Elements * Matrix_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Input_Vectors	In	M-element vector
Matrix	Input_Matrices	In	M x N input matrix

3.3.6.2.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Output_Vectors	N/A	Vector being calculated and returned

3.3.6.2.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
function Vector_Matrix_Multiply_Restricted
            (Vector : Input_Vectors;
             Matrix : Input_Matrices) return Output_Vectors is

-- ---------------------
-- --declaration section
-- ---------------------

   Answer : Output_Vectors := (others => 0.0);

-----------------------------------------------
--begin function Vector_Matrix_Multiply_Restricted
-----------------------------------------------

begin

   N_Loop:
      for N in Indices2 loop

         M_Loop:
            for M in Indices1 loop

               Answer(N) := Answer(N) + Vector(M) * Matrix(M,N);

            end loop M_Loop;

      end loop N_Loop;

   return Answer;

end Vector_Matrix_Multiply_Restricted;
```

3.3.6.2.10.7.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.7.8 LIMITATIONS

None.

```
package body General_Vector_Matrix_Algebra is

   package body Vector_Operations_Unconstrained is separate;

   package body Vector_Operations_Constrained is separate;

   package body Matrix_Operations_Unconstrained is separate;

   package body Matrix_Operations_Constrained is separate;

   package body Dynamically_Sparse_Matrix_Operations_Unconstrained is separate;

   package body Dynamically_Sparse_Matrix_Operations_Constrained is separate;

   package body Symmetric_Half_Storage_Matrix_Operations is separate;

   package body Symmetric_Full_Storage_Matrix_Operations_Unconstrained is separate;

   package body Symmetric_Full_Storage_Matrix_Operations_Constrained is separate;

   package body Diagonal_Matrix_Operations is separate;

   package body Vector_Scalar_Operations_Unconstrained is separate;

   package body Vector_Scalar_Operations_Constrained is separate;

   package body Matrix_Scalar_Operations_Unconstrained is separate;

   package body Matrix_Scalar_Operations_Constrained is separate;

   package body Diagonal_Matrix_Scalar_Operations is separate;

   package body Matrix_Vector_Multiply_Unrestricted is separate;

   function Matrix_Vector_Multiply_Restricted
               (Matrix : Input_Matrices;
                Vector : Input_Vectors) return Output_Vectors is separate;

   package body Vector_Vector_Transpose_Multiply_Unrestricted is separate;

   function Vector_Vector_Transpose_Multiply_Restricted
               (Left  : Left_Vectors ;
                Right : Right_Vectors) return Matrices is separate;

   package body Matrix_Matrix_Multiply_Unrestricted is separate;

   function Matrix_Matrix_Multiply_Restricted
               (Left  : Left_Matrices;
                Right : Right_Matrices) return Output_Matrices is separate;

   package body Matrix_Matrix_Transpose_Multiply_Unrestricted is separate;

   function Matrix_Matrix_Transpose_Multiply_Restricted
               (Left  : Left_Matrices;
                Right : Right_Matrices) return Output_Matrices is separate;

   package body Dot_Product_Operations_Unrestricted is separate;
```

```
   function Dot_Product_Operations_Restricted
               (Left  : Left_Vectors;
                Right : Right_Vectors)
               return Result_Elements is separate;

   package body Diagonal_Full_Matrix_Add_Unrestricted is separate;

   function Diagonal_Full_Matrix_Add_Restricted
               (D_Matrix : Diagonal_Matrices;
                F_Matrix : Full_Matrices) return Full_Matrices is separate;

   package body Vector_Matrix_Multiply_Unrestricted is separate;

   function Vector_Matrix_Multiply_Restricted
               (Vector : Input_Vectors;
                Matrix : Input_Matrices) return Output_Vectors is separate;

   package body Aba_Trans_Dynam_Sparse_Matrix_Sq_Matrix is separate;

   package body Aba_Trans_Vector_Sq_Matrix is separate;

   package body Aba_Trans_Vector_Scalar is separate;

   package body Column_Matrix_Operations is separate;

end General_Vector_Matrix_Algebra;
```

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Operations_Unconstrained is

pragma PAGE;
   function "+" (Left  : Vectors;
                 Right : Vectors) return Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Vectors(Left'range);
      L_Index : Indices;
      R_Index : Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --make sure lengths of input vectors are the same
      if Left'LENGTH = Right'LENGTH then

         L_Index := Left'FIRST;
         R_Index := Right'FIRST;

         Process:
            loop

               Answer(L_Index) := Left(L_Index) + Right(R_Index);

               exit Process when L_Index = Left'LAST;

               L_Index := Indices'SUCC(L_Index);
               R_Index := Indices'SUCC(R_Index);

            end loop Process;

      else

--       --dimensions of vectors are incompatible
         raise Dimension_Error;

      end if;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Left  : Vectors;
                 Right : Vectors) return Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------
```

```
      Answer   : Vectors(Left'range);
      L_Index  : Indices;
      R_Index  : Indices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

--    --make sure lengths of the input vectors are the same
      if Left'LENGTH = Right'LENGTH then

         L_Index := Left'FIRST;
         R_Index := Right'FIRST;

         Process:
            loop

               Answer(L_Index) := Left(L_Index) - Right(R_Index);

               exit Process when L_Index = Left'LAST;

               L_Index := Indices'SUCC(L_Index);
               R_Index := Indices'SUCC(R_Index);

            end loop Process;

      else

--       --dimensions of vectors are incompatible
         raise Dimension_Error;

      end if;

      return Answer;

   end "-";

pragma PAGE;
   function Vector_Length (Input : Vectors) return Vector_Elements is

--    ----------------------
--    --declaration section-
--    ----------------------

      Temp   : Vector_Elements_Squared;

-- ------------------------------
-- --begin function Vector_Length
-- ------------------------------

   begin

      Temp := 0.0;
```

```
      Process:
         for INDEX in Input'range loop
            Temp := Temp +
                    Input(INDEX) * Input(INDEX);
         end loop Process;

      return Sqrt(Temp);

   end Vector_Length;

pragma PAGE;
   function Dot_Product (Left  : Vectors;
                         Right : Vectors) return Vector_Elements_Squared is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Vector_Elements_Squared;
      L_Index  : Indices;
      R_Index  : Indices;

-- ----------------------------
-- --begin function Dot_Product
-- ----------------------------

   begin

--    --make sure lengths of the input vectors are the same
      if Left'LENGTH = Right'LENGTH then

         Answer := 0.0;
         L_Index := Left'FIRST;
         R_Index := Right'FIRST;

         Process:
            loop

               Answer := Answer + Left(L_Index) * Right(R_Index);

               exit Process when L_Index = Left'LAST;

               L_Index := Indices'SUCC(L_Index);
               R_Index := Indices'SUCC(R_Index);

            end loop Process;

      else

--       --dimensions of vectors are incompatible
         raise Dimension_Error;

      end if;

      return Answer;

   end Dot_Product;
```

```
end Vector_Operations_Unconstrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Operations_Unconstrained is

pragma PAGE;
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--     ----------------------
--     --declaration section-
--     ----------------------

       Answer   : Matrices(Left'range(1), Left'range(2));
       L_Col    : Col_Indices;
       L_Row    : Row_Indices;
       R_Col    : Col_Indices;
       R_Row    : Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--     --make sure the dimensions of the matrices are compatible
       if Left'LENGTH(1) = Right'LENGTH(1) and
          Left'LENGTH(2) = Right'LENGTH(2) then

          L_Row := Left'FIRST(1);
          R_Row := Right'FIRST(1);
          Row_Loop:
             loop

                L_Col := Left'FIRST(2);
                R_Col := Right'FIRST(2);
                Col_Loop:
                   loop

                      Answer(L_Row, L_Col) := Left(L_Row, L_Col) +
                                              Right(R_Row, R_Col);

                      exit Col_Loop when L_Col = Left'LAST(2);
                      L_Col := Col_Indices'SUCC(L_Col);
                      R_Col := Col_Indices'SUCC(R_Col);

                   end loop Col_Loop;

                exit Row_Loop when L_Row = Left'LAST(1);
                L_Row := Row_Indices'SUCC(L_Row);
                R_Row := Row_Indices'SUCC(R_Row);

             end loop Row_Loop;

       else

--        --input matrices have incompatible dimensions
          raise Dimension_Error;
```

```
      end if;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Matrices(Left'range(1), Left'range(2));
      L_Col    : Col_Indices;
      L_Row    : Row_Indices;
      R_Col    : Col_Indices;
      R_Row    : Row_Indices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

--    --make sure matrix dimensions are compatible
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then

         L_Row := Left'FIRST(1);
         R_Row := Right'FIRST(1);
         Row_Loop:
            loop

               L_Col := Left'FIRST(2);
               R_Col := Right'FIRST(2);
               Col_Loop:
                  loop

                     Answer(L_Row, L_Col) := Left(L_Row, L_Col) -
                                             Right(R_Row, R_Col);

                     exit Col_Loop when L_Col = Left'LAST(2);
                     L_Col := Col_Indices'SUCC(L_Col);
                     R_Col := Col_Indices'SUCC(R_Col);

                  end loop Col_Loop;

               exit Row_Loop when L_Row = Left'LAST(1);
               L_Row := Row_Indices'SUCC(L_Row);
               R_Row := Row_Indices'SUCC(R_Row);

            end loop Row_Loop;

      else
```

```
--          --input matrices have incompatible dimensions
            raise Dimension_Error;

         end if;

         return Answer;

      end "-";

   pragma PAGE;
      function "+" (Matrix : Matrices;
                    Addend : Elements) return Matrices is

   --    ----------------------
   --    --declaration section-
   --    ----------------------

         Answer : Matrices(Matrix'range(1), Matrix'range(2));

   -- --------------------
   -- --begin function "+"
   -- --------------------

      begin

         Row_Loop:
            for Row in Matrix'range(1) loop
               Col_Loop:
                  for COL in Matrix'range(2) loop
                     Answer(Row, COL) := Matrix(Row, COL) + Addend;
                  end loop Col_Loop;
            end loop Row_Loop;

         return Answer;

      end "+";

   pragma PAGE;
      function "-" (Matrix     : Matrices;
                    Subtrahend : Elements) return Matrices is

   --    ----------------------
   --    --declaration section-
   --    ----------------------

         Answer : Matrices(Matrix'range(1), Matrix'range(2));

   -- --------------------
   -- --begin function "-"
   -- --------------------

      begin

         Row_Loop:
            for Row in Matrix'range(1) loop
               Col_Loop:
                  for COL in Matrix'range(2) loop
```

```
                Answer(Row, COL) := Matrix(Row, COL) - Subtrahend;
             end loop Col_Loop;
          end loop Row_Loop;

       return Answer;

   end "-";

pragma PAGE;
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--     --------------------
--     --declaration section
--     --------------------

       Col_Marker : Col_Indices;
       Row        : Row_Indices;

-- ---------------------------------------
-- --begin function Set_To_Identity_Matrix
-- ---------------------------------------

   begin

--     --make sure input matrix is a square matrix
       if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

          Matrix := (others => (others => 0.0));

          Row        := Matrix'FIRST(1);
          Col_Marker := Matrix'FIRST(2);
          Row_Loop:
             loop

--              --set diagonal element equal to 1
                Matrix(Row, Col_Marker) := 1.0;

                exit Row_Loop when Row = Matrix'LAST(1);
                Row        := Row_Indices'SUCC(Row);
                Col_Marker := Col_Indices'SUCC(Col_Marker);

             end loop Row_Loop;

       else

--        --do not have a square matrix
          raise Dimension_Error;

       end if;

   end Set_To_Identity_Matrix;

pragma PAGE;
   procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

   begin
```

```
      Matrix := (others => (others => 0.0));

   end Set_To_Zero_Matrix;

pragma PAGE;
   function "*" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer  : Matrices(Left'range(1), Right'range(2));
       M       : Row_Indices;
       N_Left  : Col_Indices;
       N_Right : Row_Indices;
       P       : Col_Indices;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

--     --make sure dimensions are compatible
       if Left'LENGTH(2) = Right'LENGTH(1) then

          M := Left'FIRST(1);
          M_Loop:
             loop

                P := Right'FIRST(2);
                P_Loop:
                   loop

                      Answer(M,P) := 0.0;

                      N_Left  := Left'FIRST(2);
                      N_Right := Right'FIRST(1);
                      N_Loop:
                         loop

                            Answer(M,P) := Answer(M,P) +
                                           Left(M,N_Left) * Right(N_Right,P);

                            exit N_Loop when N_Left = Left'LAST(2);
                            N_Left  := Col_Indices'SUCC(N_Left);
                            N_Right := Row_Indices'SUCC(N_Right);

                         end loop N_Loop;

                      exit P_Loop when P = Right'LAST(2);
                      P := Col_Indices'SUCC(P);

                   end loop P_Loop;

                exit M_Loop when M = Left'LAST(1);
```

```
                M := Row_Indices'SUCC(M);

            end loop M_Loop;

        else

--          --dimensions are incompatible
            raise Dimension_Error;

        end if;

        return Answer;

    end "*";

end Matrix_Operations_Unconstrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Dynamically_Sparse_Matrix_Operations_Unconstrained is

pragma PAGE;
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--     ---------------------
--     --declaration section
--     ---------------------

       Col_Marker : Col_Indices;
       Row        : Row_Indices;

-- ----------------------------------------
-- --begin procedure Set_to_Identity_Matrix
-- ----------------------------------------

   begin

--     --make sure input matrix is a square matrix
       if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

          Matrix := (others => (others => 0.0));

          Row        := Matrix'FIRST(1);
          Col_Marker := Matrix'FIRST(2);
          Row_Loop:
             loop

--              --set diagonal element equal to 1.0
                Matrix(Row, Col_Marker) := 1.0;

                exit Row_Loop when Row = Matrix'LAST(1);
                Row        := Row_Indices'SUCC(Row);
                Col_Marker := Col_Indices'SUCC(Col_Marker);

             end loop Row_Loop;

       else

          raise Dimension_Error;

       end if;

   end Set_To_Identity_Matrix;

pragma PAGE;
   procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

   begin

      Matrix := (others => (others => 0.0));

   end Set_To_Zero_Matrix;

pragma PAGE;
   function Add_To_Identity (Input : Matrices) return Matrices is
```

```
--    ---------------------
--    --declaration section
--    ---------------------

      Answer     : Matrices(Input'range(1),Input'range(2));
      Col_Marker : Col_Indices;
      Row        : Row_Indices;

-- ---------------------------------
-- --begin procedure Add_to_Identity
-- ---------------------------------

   begin

--    --make sure input is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then

         Answer := Input;

--       --add "identity" values to diagonal elements
         Row        := Input'FIRST(1);
         Col_Marker := Input'FIRST(2);
         Row_Loop:
            loop

               if Answer(Row, Col_Marker) /= 0.0 then
                  Answer(Row, Col_Marker) := Answer(Row, Col_Marker) + 1.0;
               else
                  Answer(Row, Col_Marker) := 1.0;
               end if;

               exit Row_Loop when Row = Input'LAST(1);
               Row        := Row_Indices'SUCC(Row);
               Col_Marker := Col_Indices'SUCC(Col_Marker);

            end loop Row_Loop;

      else

         raise Dimension_Error;

      end if;

      return Answer;

   end Add_To_Identity;

pragma PAGE;
   function Subtract_From_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer     : Matrices(Input'range(1),Input'range(2));
      Col_Marker : Col_Indices;
```

```
      Row         : Row_Indices;

-- ----------------------------------------
-- --begin procedure Subtract_From_Identity
-- ----------------------------------------

   begin

--    --make sure input is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then

         Row        := Input'FIRST(1);
         Col_Marker := Input'FIRST(2);
         Row_Loop:
            loop

               Col_Loop:
                  for COL in Input'range(2) loop
                     if Input(Row,COL) /= 0.0 then
                        Answer(Row,COL) := - Input(Row,COL);
                     else
                        Answer(Row,COL) := 0.0;
                     end if;
                  end loop Col_Loop;

               if Answer(Row, Col_Marker) /= 0.0 then
                  Answer(Row, Col_Marker) := Answer(Row, Col_Marker) + 1.0;
               else
                  Answer(Row, Col_Marker) := 1.0;
               end if;

               exit Row_Loop when Row = Input'LAST(1);
               Row        := Row_Indices'SUCC(Row);
               Col_Marker := Col_Indices'SUCC(Col_Marker);

            end loop Row_Loop;

      else

         raise Dimension_Error;

      end if;

      return Answer;

   end Subtract_From_Identity;

pragma PAGE;
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Matrices(Left'range(1), Left'range(2));
      L_Col  : Col_Indices;
```

```
      L_Row  : Row_Indices;
      R_Col  : Col_Indices;
      R_Row  : Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --make sure have compatible dimensions
      if Left'LENGTH(1) = Right'LENGTH(1) and then
         Left'LENGTH(2) = Right'LENGTH(2) then

         L_Row := Left'FIRST(1);
         R_Row := Right'FIRST(1);
         Row_Loop:
            loop

               L_Col := Left'FIRST(2);
               R_Col := Right'FIRST(2);
               Col_Loop:
                  loop

                     if Left(L_Row, L_Col) = 0.0 then
                        if Right(R_Row, R_Col) = 0.0 then
                           Answer(L_Row, L_Col) := 0.0;
                        else
                           Answer(L_Row, L_Col) := Right(R_Row, R_Col);
                        end if;
                     elsif Right(R_Row, R_Col) = 0.0 then
                        Answer(L_Row, L_Col) := Left(L_Row, L_Col);
                     else
                        Answer(L_Row, L_Col) := Left(L_Row, L_Col) +
                                                Right(R_Row, R_Col);
                     end if;

                     exit Col_Loop when L_Col = Left'LAST(2);
                     L_Col := Col_Indices'SUCC(L_Col);
                     R_Col := Col_Indices'SUCC(R_Col);

                  end loop Col_Loop;

               exit Row_Loop when L_Row = Left'LAST(1);
               L_Row := Row_Indices'SUCC(L_Row);
               R_Row := Row_Indices'SUCC(R_Row);

            end loop Row_Loop;

      else

         raise Dimension_Error;

      end if;

      return Answer;
```

```
   end "+";

pragma PAGE;
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Matrices(Left'range(1), Left'range(2));
      L_Col  : Col_Indices;
      L_Row  : Row_Indices;
      R_Col  : Col_Indices;
      R_Row  : Row_Indices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

--    --make sure have compatible dimensions
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then

         L_Row := Left'FIRST(1);
         R_Row := Right'FIRST(1);
         Row_Loop:
            loop

               L_Col := Left'FIRST(2);
               R_Col := Right'FIRST(2);
               Col_Loop:
                  loop

                     if Left(L_Row, L_Col) = 0.0 then
                        if Right(R_Row, R_Col) = 0.0 then
                           Answer(L_Row, L_Col) := 0.0;
                        else
                           Answer(L_Row, L_Col) := - Right(R_Row, R_Col);
                        end if;
                     elsif Right(R_Row, R_Col) = 0.0 then
                        Answer(L_Row, L_Col) := Left(L_Row, L_Col);
                     else
                        Answer(L_Row, L_Col) := Left(L_Row, L_Col) -
                                                Right(R_Row, R_Col);
                     end if;

                     exit Col_Loop when L_Col = Left'LAST(2);
                     L_Col := Col_Indices'SUCC(L_Col);
                     R_Col := Col_Indices'SUCC(R_Col);

                  end loop Col_Loop;

               exit Row_Loop when L_Row = Left'LAST(1);
               L_Row := Row_Indices'SUCC(L_Row);
```

```
            R_Row := Row_Indices'SUCC(R_Row);
         end loop Row_Loop;
      else
         raise Dimension_Error;
      end if;
      return Answer;
   end "-";
end Dynamically_Sparse_Matrix_Operations_Unconstrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Symmetric_Half_Storage_Matrix_Operations is

-- --------------------
-- --local declarations
-- --------------------

   type Col_Index_Arrays is array(Col_Indices) of NATURAL;
   type Row_Index_Arrays is array(Row_Indices) of NATURAL;

   Col_Offset : Col_Index_Arrays;
   Row_Marker : Row_Index_Arrays;

-- --this object is initially only zeroed out; the 1.0 values will be assigned
-- --to the diagonal elements during package initialization
   Local_Identity_Matrix : Matrices := (others => 0.0);

   Local_Zero_Matrix     : constant Matrices := (others => 0.0);

pragma PAGE;
   function Swap_Col (Row : Row_Indices) return Col_Indices is
   begin
      return Col_Indices'VAL(Row_Indices'POS(Row) -
                             Row_Indices'POS(Row_Indices'FIRST) +
                             Col_Indices'POS(Col_Indices'FIRST));
   end Swap_Col;

pragma PAGE;
   function Swap_Row (COL : Col_Indices) return Row_Indices is
   begin
      return Row_Indices'VAL(Col_Indices'POS(COL) -
                             Col_Indices'POS(Col_Indices'FIRST) +
                             Row_Indices'POS(Row_Indices'FIRST));
   end Swap_Row;

pragma PAGE;
   procedure Initialize (Row_Slice : in     Row_Slices;
                         Row       : in     Row_Indices;
                         Matrix    :    out Matrices) is

--    ---------------------
--    --declaration section
--    ---------------------

      INDEX     : Col_Indices;
      Marker    : POSITIVE;
      Stop_Here : POSITIVE;

-- ----------------------------
-- --begin procedure Initialize
-- ----------------------------

   begin

      INDEX     := Col_Indices'FIRST;
      Marker    := Row_Marker(Row);
      Stop_Here := Marker + Col_Offset(Swap_Col(Row));
```

```
      Process:
         loop

            Matrix(Marker) := Row_Slice(INDEX);

            exit Process when Marker = Stop_Here;
            INDEX  := Col_Indices'SUCC(INDEX);
            Marker := Marker + 1;

         end loop Process;

   end Initialize;

pragma PAGE;
   function Identity_Matrix return Matrices is

   begin

      return Local_Identity_Matrix;

   end Identity_Matrix;

pragma PAGE;
   function Zero_Matrix return Matrices is

   begin

      return Local_Zero_Matrix;

   end Zero_Matrix;

pragma PAGE;
   procedure Change_Element (New_Value : in     Elements;
                             Row       : in     Row_Indices;
                             COL       : in     Col_Indices;
                             Matrix    :    out Matrices) is

   begin

--    --determine which half of the matrix is being referenced
      if Row_Indices'POS(Row) - Row_Indices'POS(Row_Indices'FIRST) >=
         Col_Indices'POS(COL) - Col_Indices'POS(Col_Indices'FIRST) then

--       --looking at bottom half of array
         Matrix(Row_Marker(Row) + Col_Offset(COL)) := New_Value;

      else

--       --looking at top half; need to switch to bottom half
         Matrix(Row_Marker(Swap_Row(COL)) +
                Col_Offset(Swap_Col(Row))) := New_Value;

      end if;

   end Change_Element;

pragma PAGE;
```

```
   function Retrieve_Element (Matrix : Matrices;
                              Row    : Row_Indices;
                              COL    : Col_Indices) return Elements is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer : Elements;

-- ---------------------------------
-- --begin function Retrieve_Element
-- ---------------------------------

   begin

--     --determine which half of the array is being referenced
       if Row_Indices'POS(Row) - Row_Indices'POS(Row_Indices'FIRST) >=
          Col_Indices'POS(COL) - Col_Indices'POS(Col_Indices'FIRST) then

--        --already looking at the bottom half of the array
          Answer := Matrix(Row_Marker(Row) + Col_Offset(COL));

       else

--        --looking at the top half; need to switch to bottom half
          Answer := Matrix(Row_Marker(Swap_Row(COL)) +
                           Col_Offset(Swap_Col(Row)));

       end if;

       return Answer;

   end Retrieve_Element;

pragma PAGE;
   function Row_Slice (Matrix : Matrices;
                       Row    : Row_Indices) return Row_Slices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer : Row_Slices;

-- --------------------------
-- --begin function Row_Slice
-- --------------------------

   begin

--     --retrieve row elements in bottom half of array
       Bottom_Loop:
          for COL in Col_Indices'FIRST .. Swap_Col(Row) loop
             Answer(COL) := Matrix(Row_Marker(Row) + Col_Offset(COL));
          end loop Bottom_Loop;
```

```
--       --retrieve row elements in top half of array, if there are any
         if Row /= Row_Indices'LAST then
            Top_Loop:
               for COL in Col_Indices'SUCC(Swap_Col(Row)) .. Col_Indices'LAST loop
                  Answer(COL) := Matrix(Row_Marker(Swap_Row(COL)) +
                                        Col_Offset(Swap_Col(Row)));
               end loop Top_Loop;
         end if;

         return Answer;

      end Row_Slice;

pragma PAGE;
      function Column_Slice (Matrix : Matrices;
                             COL    : Col_Indices) return Col_Slices is

--       ---------------------
--       --declaration section
--       ---------------------

         Answer : Col_Slices;

-- ----------------------------
-- --begin function Column_Slice
-- ----------------------------

      begin

--       --retrieve column elements contained in bottom half of array
         Bottom_Loop:
            for Row in Swap_Row(COL) .. Row_Indices'LAST loop
               Answer(Row) := Matrix(Row_Marker(Row) + Col_Offset(COL));
            end loop Bottom_Loop;

--       --retrieve column elements contained in top half of array, if any
         if COL /= Col_Indices'FIRST then
            Top_Loop:
               for Row in Row_Indices'FIRST .. Row_Indices'PRED(Swap_Row(COL)) loop
                  Answer(Row) := Matrix(Row_Marker(Swap_Row(COL)) +
                                        Col_Offset(Swap_Col(Row)));
               end loop Top_Loop;
         end if;

         return Answer;

      end Column_Slice;

pragma PAGE;
      function Add_To_Identity (Input : Matrices) return Matrices is

--       ---------------------
--       --declaration section
--       ---------------------

         Answer : Matrices;
```

```
-- ------------------------------
-- --begin function Add_To_Identity
-- ------------------------------

   begin

--       --do straight assignment of all elements and then add in the
--       --identity matrix

         Answer := Input;

--       --all diagonal elements, except for the last one, are located one
--       --entry before the starting location of the next row
         Add_Identity_Loop:
            for INDEX in Row_Indices'SUCC(Row_Indices'FIRST) ..
                         Row_Indices'LAST loop
               Answer(Row_Marker(INDEX) - 1) := Answer(Row_Marker(INDEX)-1) + 1.0;
            end loop Add_Identity_Loop;

--       --handle last diagonal element
         Answer(Entry_Count) := Answer(Entry_Count) + 1.0;

         return Answer;

   end Add_To_Identity;

pragma PAGE;
   function Subtract_From_Identity (Input : Matrices) return Matrices is

--       --------------------
--       --declaration section
--       --------------------

         Answer : Matrices;

-- ---------------------------------------
-- --begin function Subtract_from_Identity
-- ---------------------------------------

   begin

--       --subtract Input from a zero matrix and then add it to an identity matrix

         Subtract_Loop:
            for INDEX in 1..Entry_Count loop
               Answer(INDEX) := -Input(INDEX);
            end loop Subtract_Loop;

--       --all diagonal elements, except for the last one, are located one
--       --entry before the starting location of the next row
         Add_Identity_Loop:
            for INDEX in Row_Indices'SUCC(Row_Indices'FIRST) ..
                         Row_Indices'LAST loop
               Answer(Row_Marker(INDEX) - 1) := Answer(Row_Marker(INDEX)-1) + 1.0;
            end loop Add_Identity_Loop;

--       --handle last diagonal element
```

```
      Answer(Entry_Count) := Answer(Entry_Count) + 1.0;

      return Answer;

   end Subtract_From_Identity;

pragma PAGE;
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Process:
         for INDEX in 1..Entry_Count loop
            Answer(INDEX) := Left(INDEX) + Right(INDEX);
         end loop Process;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Matrices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Process:
         for INDEX in 1 .. Entry_Count loop
            Answer(INDEX) := Left(INDEX) - Right(INDEX);
         end loop Process;

      return Answer;

   end "-";

pragma PAGE;
```

```
-----------------------------------------------
--begin processing for Symmetric_Half_Storage_
--Matrix_Operations package body
-----------------------------------------------

begin

   Init_Block:
      declare

         COUNT              : NATURAL;
         Offset             : NATURAL;
         Row_Starting_Point : NATURAL;

      begin

--       --make sure lengths of row and col indices are the same
         if Row_Slices'LENGTH /= Col_Slices'LENGTH then

            raise Dimension_Error;

         else

--          ------------------------------------------------------------------
--          --initialize row marker identity matrix arrays;
--          --all diagonal elements, except for the last one, which require
--          --   a value of 1 for the identity matrix are located one entry
--          --   before the starting location of the next row
--          ------------------------------------------------------------------

--          --handle first row marker entry to simplify initialization of
--          --the identity matrix --(NOTE: count implicitly equals 0)
            Row_Marker(Row_Indices'FIRST) := 1;

            COUNT := 1;
            Row_Marker_And_Identity_Matrix_Init_Loop:
               for INDEX in Row_Indices'SUCC(Row_Indices'FIRST) ..
                            Row_Indices'LAST loop

                  Row_Starting_Point := (COUNT * (COUNT+1) / 2) + 1;

                  Row_Marker(INDEX) := Row_Starting_Point;
                  Local_Identity_Matrix(Row_Starting_Point-1) := 1.0;

                  COUNT := COUNT + 1;

               end loop Row_Marker_And_Identity_Matrix_Init_Loop;

--          --initialize last diagonal element
            Local_Identity_Matrix(Entry_Count) := 1.0;

--          --------------------------------
--          --initialize column offset array
--          --------------------------------

            Offset := 0;
            Col_Marker_Init_Loop:
```

```
                for INDEX in Col_Indices loop
                   Col_Offset(INDEX) := Offset;
                   Offset := Offset + 1;
                end loop Col_Marker_Init_Loop;

            end if;

         end Init_Block;

end Symmetric_Half_Storage_Matrix_Operations;
```

```
separate (General_Vector_Matrix_Algebra)
package body Symmetric_Full_Storage_Matrix_Operations_Unconstrained is

pragma PAGE;
   procedure Change_Element (New_Value : in     Elements;
                             Row       : in     Row_Indices;
                             COL       : in     Col_Indices;
                             Matrix    : in out Matrices) is

--    ----------------------
--    --declaration section-
--    ----------------------

      S_Col : Col_Indices;
      S_Row : Row_Indices;

-- ---------------------------------
-- --begin procedure Change_Element-
-- ---------------------------------

   begin

--    --make sure you have a square matrix
      if Matrix'LENGTH(1) /= Matrix'LENGTH(2) then

         raise Dimension_Error;

--    --make sure row and col are within bounds
      elsif not (Row in Matrix'range(1)  and
                 COL in Matrix'range(2)) then

         raise Invalid_Index;

      else

--       --everything is okay

         S_Col := Col_Indices'VAL(Row_Indices'POS(Row) -
                                  Row_Indices'POS(Matrix'FIRST(1)) +
                                  Col_Indices'POS(Matrix'FIRST(2)));

         S_Row := Row_Indices'VAL(Col_Indices'POS(COL) -
                                  Col_Indices'POS(Matrix'FIRST(2)) +
                                  Row_Indices'POS(Matrix'FIRST(1)));

         Matrix(Row, COL)     := New_Value;
         Matrix(S_Row, S_Col) := New_Value;

      end if;

   end Change_Element;

pragma PAGE;
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

-- ----------------------
-- --declaration section-
```

```
-- ----------------------

   COL : Col_Indices;
   Row : Row_Indices;

-- ---------------------------------
-- --begin procedure Set_to_Identity-
-- ---------------------------------

   begin

--     --make sure input matrix is a square matrix
       if Matrix'LENGTH(1) = Matrix'LENGTH(2) then


          Matrix := (others => (others => 0.0));

          Row := Matrix'FIRST(1);
          COL := Matrix'FIRST(2);
          Row_Loop:
             loop

--              --set diagonal element equal to
                Matrix(Row, COL) := 1.0;

                exit Row_Loop when Row = Matrix'LAST(1);
                Row := Row_Indices'SUCC(Row);
                COL := Col_Indices'SUCC(COL);

             end loop Row_Loop;

       else

--        --do not have a square matrix
          raise Dimension_Error;

       end if;

   end Set_To_Identity_Matrix;

pragma PAGE;
   procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

   begin

      Matrix := (others => (others => 0.0));

   end Set_To_Zero_Matrix;

pragma PAGE;
   function Add_To_Identity (Input : Matrices) return Matrices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer      : Matrices(Input'range(1), Input'range(2));
```

```
      COL         : Col_Indices;
      Row         : Row_Indices;

-- -------------------------------
-- --begin function Add_to_Identity
-- -------------------------------

   begin

--    --make sure input matrix is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then

         Answer := Input;

         Row := Input'FIRST(1);
         COL := Input'FIRST(2);
         Access_Diagonal_Elements:
            loop

               Answer(Row,COL) := Answer(Row,COL) + 1.0;

               exit Access_Diagonal_Elements when Row = Input'LAST(1);
               Row := Row_Indices'SUCC(Row);
               COL := Col_Indices'SUCC(COL);

            end loop Access_Diagonal_Elements;

      else

--       --do not have a square matrix
         raise Dimension_Error;

      end if;

      return Answer;

   end Add_To_Identity;

pragma PAGE;
   function Subtract_From_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer    : Matrices(Input'range(1), Input'range(2));
      COL       : Col_Indices;
      Col_Count : POSITIVE;
      Row       : Row_Indices;
      Row_Count : POSITIVE;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- ---------------------------------------
-- --begin function Subtract_from_Identity
-- ---------------------------------------
```

```
   begin

--     --make sure input matrix is a square matrix
       if Input'LENGTH(1) = Input'LENGTH(2) then

--         --will subtract input matrix from an identity matrix by first
--         --subtracting all elements from 0.0 and then adding 1.0 to the
--         --diagonal elements;
--         --when doing the subtraction, will only calculate the remainder
--         --for the elements in the bottom half of the matrix and will simply
--         --do assignments for the symmetric elements in the top half of the
--         --matrix

           Row_Count := 1;

--         --S_Col will go across the columns as Row goes down the rows;
--         --will mark column containing the diagonal element for this row
           Row   := Input'FIRST(1);
           S_Col := Input'FIRST(2);
           Do_Every_Row:
              loop

                 Col_Count := 1;

--               --S_Row will go down the rows as Col goes across the columns;
--               --when paired with S_Col will mark the symmetric counterpart
--               --to the element being referenced in the bottom half of the
--               --matrix
                 COL       := Input'FIRST(2);
                 S_Row     := Input'FIRST(1);
                 Subtract_Elements_From_Zero:
                    loop

--                     --perform subtraction on element in bottom half of matrix
                       Answer(Row,COL) := - Input(Row,COL);

--                     --exit loop after diagonal element has been reached
                       exit Subtract_Elements_From_Zero when Col_Count =
                                                             Row_Count;

--                     --assign values to symmetric elements in top half of matrix
--                     --(done after check for diagonal, since diagonal elements
--                     --  don't have a symmetric counterpart)
                       Answer(S_Row,S_Col) := Answer(Row,COL);

--                     --increment variables
                       Col_Count := Col_Count + 1;
                       COL       := Col_Indices'SUCC(COL);
                       S_Row     := Row_Indices'SUCC(S_Row);

                    end loop Subtract_Elements_From_Zero;

--               --add one to the diagonal element
                 Answer(Row, COL) := Answer(Row, S_Col) + 1.0;

                 exit Do_Every_Row when Row_Count = Input'LENGTH(1);
                 Row_Count := Row_Count + 1;
```

```
                Row       := Row_Indices'SUCC(Row);
                S_Col     := Col_Indices'SUCC(S_Col);

             end loop Do_Every_Row;

      else

         raise Dimension_Error;

      end if;

      return Answer;

   end Subtract_From_Identity;

pragma PAGE;
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    --------------------
--    --declaration section
--    --------------------

      Answer    : Matrices(Left'range(1), Left'range(2));
      Col_Count : POSITIVE;
      Row_Count : POSITIVE;
      L_Col     : Col_Indices;
      L_Row     : Row_Indices;
      R_Col     : Col_Indices;
      R_Row     : Row_Indices;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --make sure both input matrices are square matrices of the same size
      if Left'LENGTH(1)  = Left'LENGTH(2) and
         Left'LENGTH(1)  = Right'LENGTH(1) and
         Right'LENGTH(1) = Right'LENGTH(2) then

--       --addition calculations will only be carried out on the bottom half
--       --of the input matrices followed by assignments to the symmetric
--       --elements in the top half of the matrix

         Row_Count := 1;

--       --as L_Row goes down the rows, S_Col will go across the columns
         L_Row     := Left'FIRST(1);
         S_Col     := Left'FIRST(2);

         R_Row     := Right'FIRST(1);
         Do_All_Rows:
            loop
```

```
                Col_Count := 1;

--              --as L_Col goes across the columns, S_Row will go down the rows
                L_Col := Left'FIRST(2);
                S_Row := Left'FIRST(1);

                R_Col := Right'FIRST(2);
                Add_Bottom_Half_Elements:
                   loop

                      Answer(L_Row,L_Col) := Left(L_Row, L_Col) +
                                             Right(R_Row, R_Col);

--                    --exit when diagonal element has been reached
                      exit Add_Bottom_Half_Elements when Col_Count = Row_Count;

--                    --assign value to symmetric element in top half of matrix
--                    --(do this after exit since diagonal elements don't have
--                    --  a corresponding symmetric element)
                      Answer(S_Row,S_Col) := Answer(L_Row,L_Col);

--                    --increment values
                      Col_Count := Col_Count + 1;
                      L_Col     := Col_Indices'SUCC(L_Col);
                      S_Row     := Row_Indices'SUCC(S_Row);
                      R_Col     := Col_Indices'SUCC(R_Col);

                   end loop Add_Bottom_Half_Elements;

                exit Do_All_Rows when Row_Count = Left'LENGTH(1);
                Row_Count := Row_Count + 1;
                L_Row     := Row_Indices'SUCC(L_Row);
                S_Col     := Col_Indices'SUCC(S_Col);
                R_Row     := Row_Indices'SUCC(R_Row);

             end loop Do_All_Rows;

          else

             raise Dimension_Error;

          end if;

          return Answer;

       end "+";

    pragma PAGE;
       function "-" (Left  : Matrices;
                     Right : Matrices) return Matrices is

--        ---------------------
--        --declaration section
--        ---------------------

          Answer    : Matrices(Left'range(1), Left'range(2));
```

```
      Col_Count : POSITIVE;
      Row_Count : POSITIVE;
      L_Col     : Col_Indices;
      L_Row     : Row_Indices;
      R_Col     : Col_Indices;
      R_Row     : Row_Indices;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--    --make sure both input matrices are square matrices of the same size
      if Left'LENGTH(1)  = Left'LENGTH(2) and
         Left'LENGTH(1)  = Right'LENGTH(1) and
         Right'LENGTH(1) = Right'LENGTH(2) then

--       --addition calculations will only be carried out on the bottom half
--       --of the input matrices followed by assignments to the symmetric
--       --elements in the top half of the matrix

         Row_Count := 1;

--       --as L_Row goes down the rows, S_Col will go across the columns
         L_Row     := Left'FIRST(1);
         S_Col     := Left'FIRST(2);

         R_Row     := Right'FIRST(1);
         Do_All_Rows:
            loop

               Col_Count := 1;

--             --as L_Col goes across the columns, S_Row will go down the rows
               L_Col := Left'FIRST(2);
               S_Row := Left'FIRST(1);

               R_Col := Right'FIRST(2);
               Add_Bottom_Half_Elements:
                  loop

                     Answer(L_Row,L_Col) := Left(L_Row, L_Col) -
                                            Right(R_Row, R_Col);

--                   --exit when diagonal element has been reached
                     exit Add_Bottom_Half_Elements when Col_Count = Row_Count;

--                   --assign value to symmetric element in top half of matrix
--                   --(do this after exit since diagonal elements don't have
--                   --  a corresponding symmetric element)
                     Answer(S_Row,S_Col) := Answer(L_Row,L_Col);

--                   --increment values
                     Col_Count := Col_Count + 1;
```

```
                    L_Col      := Col_Indices'SUCC(L_Col);
                    S_Row      := Row_Indices'SUCC(S_Row);
                    R_Col      := Col_Indices'SUCC(R_Col);

                end loop Add_Bottom_Half_Elements;

                exit Do_All_Rows when Row_Count = Left'LENGTH(1);
                Row_Count := Row_Count + 1;
                L_Row      := Row_Indices'SUCC(L_Row);
                S_Col      := Col_Indices'SUCC(S_Col);
                R_Row      := Row_Indices'SUCC(R_Row);

            end loop Do_All_Rows;

        else

            raise Dimension_Error;

        end if;

        return Answer;

    end "-";

end Symmetric_Full_Storage_Matrix_Operations_Unconstrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Diagonal_Matrix_Operations is

-- --------------------
-- --local declarations
-- --------------------

   type Col_Markers is array(Col_Indices) of POSITIVE;
   type Row_Markers is array(Row_Indices) of POSITIVE;

   Col_Marker : Col_Markers;
   Row_Marker : Row_Markers;

   Row_Minus_Col_Indices_Pos_First : constant INTEGER
                                     := Row_Indices'POS(Row_Indices'FIRST) -
                                        Col_Indices'POS(Col_Indices'FIRST);

   Local_Identity_Matrix : constant Diagonal_Matrices := (others => 1.0);
   Local_Zero_Matrix     : constant Diagonal_Matrices := (others => 0.0);

pragma PAGE;
   function Identity_Matrix return Diagonal_Matrices is

   begin

      return Local_Identity_Matrix;

   end Identity_Matrix;

pragma PAGE;
   function Zero_Matrix return Diagonal_Matrices is

   begin

      return Local_Zero_Matrix;

   end Zero_Matrix;

pragma PAGE;
   procedure Change_Element (New_Value : in     Elements;
                             Row       : in     Row_Indices;
                             COL       : in     Col_Indices;
                             Matrix    :    out Diagonal_Matrices) is

   begin

--    --make sure element referenced is on the diagonal
      if Row_Marker(Row) = Col_Marker(COL) then

         Matrix(Row_Marker(Row)) := New_Value;

      else

         raise Invalid_Index;

      end if;
```

```
   end Change_Element;

pragma PAGE;
   function Retrieve_Element (Matrix  : Diagonal_Matrices;
                              Row     : Row_Indices;
                              COL     : Col_Indices) return Elements is

   begin

--     --make sure (row,col) falls on the diagonal
       if Row_Marker(Row) /= Col_Marker(COL) then
          raise Invalid_Index;
       end if;

       return Matrix(Row_Marker(Row));

   end Retrieve_Element;

pragma PAGE;
   function Row_Slice (Matrix : Diagonal_Matrices;
                       Row    : Row_Indices) return Row_Slices is

--     ---------------------
--     --declaration section
--     ---------------------

       Col_Spot : Col_Indices;
       Answer   : Row_Slices;

-- --------------------------
-- --begin function Row_Slice
-- --------------------------

   begin

--     --zero out slice
       Answer := (others => 0.0);

--     --insert diagonal element
       Col_Spot := Col_Indices'VAL(Row_Indices'POS(Row) -
                                   Row_Minus_Col_Indices_Pos_First);
       Answer(Col_Spot) := Matrix(Row_Marker(Row));

       return Answer;

   end Row_Slice;

pragma PAGE;
   function Column_Slice (Matrix : Diagonal_Matrices;
                          COL    : Col_Indices) return Col_Slices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer   : Col_Slices;
       Row_Spot : Row_Indices;
```

```
-- ----------------------------
-- --begin function Column_Slice
-- ----------------------------

   begin

--      --zero out answer and then insert diagonal value
        Answer := (others => 0.0);

--      --insert diagonal value
        Row_Spot := Row_Indices'VAL(Col_Indices'POS(COL) +
                                    Row_Minus_Col_Indices_Pos_First);
        Answer(Row_Spot) := Matrix(Col_Marker(COL));

        return Answer;

   end Column_Slice;

pragma PAGE;
   function Add_To_Identity (Input : Diagonal_Matrices)
                             return Diagonal_Matrices is

--      ---------------------
--      --declaration section
--      ---------------------

        Answer : Diagonal_Matrices;

-- --------------------------------
-- --begin function Add_to_Identity
-- --------------------------------

   begin

      Process:
         for INDEX in 1..Entry_Count loop
            Answer(INDEX) := Input(INDEX) + 1.0;
         end loop Process;

      return Answer;

   end Add_To_Identity;

pragma PAGE;
   function Subtract_From_Identity (Input : Diagonal_Matrices)
                                    return Diagonal_Matrices is

--      ---------------------
--      --declaration section
--      ---------------------

        Answer : Diagonal_Matrices;

-- ---------------------------- ----------
-- --begin function Subtract_From_Identity
-- ---------------------------------------
```

```
   begin

      Process:
         for INDEX in 1..Entry_Count loop
            Answer(INDEX) := 1.0 - Input(INDEX);
         end loop Process;

      return Answer;

   end Subtract_From_Identity;

pragma PAGE;
   function "+" (Left  : Diagonal_Matrices;
                 Right : Diagonal_Matrices) return Diagonal_Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Diagonal_Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Process:
         for INDEX in 1..Entry_Count loop
            Answer(INDEX) := Left(INDEX) + Right(INDEX);
         end loop Process;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Left  : Diagonal_Matrices;
                 Right : Diagonal_Matrices) return Diagonal_Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Diagonal_Matrices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Process:
         for INDEX in 1..Entry_Count loop
            Answer(INDEX) := Left(INDEX) - Right(INDEX);
         end loop Process;
```

```
      return Answer;

   end "-";

pragma PAGE;
--------------------------------
--begin processing for Diagonal_
--Matrix_Operations package
--------------------------------
begin

   Init_Block:
      declare

         Col_Count : POSITIVE;
         Row_Count : POSITIVE;

      begin

--       --make sure lengths of indices are the same
         if Row_Slices'LENGTH = Col_Slices'LENGTH then

--          --initialize row and column marker arrays

            Row_Count := 1;
            Row_Init:
               for Row in Row_Indices loop
                  Row_Marker(Row) := Row_Count;
                  Row_Count       := Row_Count + 1;
               end loop Row_Init;

            Col_Count := 1;
            Col_Init:
               for COL in Col_Indices loop
                  Col_Marker(COL) := Col_Count;
                  Col_Count       := Col_Count + 1;
               end loop Col_Init;

         else

            raise Dimension_Error;

         end if;

      end Init_Block;

end Diagonal_Matrix_Operations;
```

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Scalar_Operations_Unconstrained is

pragma PAGE;
   function "*" (Vector     : Vectors2;
                 Multiplier : Scalars) return Vectors1 is

--     ----------------------
--     --declaration section-
--     ----------------------

       Answer  : Vectors1(Indices1'FIRST ..
                          Indices1'VAL(Vector'LENGTH-1 +
                                       Indices1'POS(Indices1'FIRST) ));
       A_Index : Indices1;
       V_Index : Indices2;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

     A_Index := Indices1'FIRST;
     V_Index := Indices2'FIRST;
     Process:
        loop

           Answer(A_Index) := Vector(V_Index) * Multiplier;

           exit Process when V_Index = Vector'LAST;
           A_Index := Indices1'SUCC(A_Index);
           V_Index := Indices2'SUCC(V_Index);

        end loop Process;

     return Answer;

   end "*";

pragma PAGE;
   function "/" (Vector  : Vectors1;
                 Divisor : Scalars) return Vectors2 is

--     ----------------------
--     --declaration section-
--     ----------------------

       Answer  : Vectors2(Indices2'FIRST ..
                          Indices2'VAL(Vector'LENGTH-1 +
                                       Indices2'POS(Indices2'FIRST) ));
       A_Index : Indices2;
       V_Index : Indices1;

-- ------------------------------------
-- --begin function Vector_Scalar_Divide
-- ------------------------------------
```

```
   begin

      A_Index := Indices2'FIRST;
      V_Index := Indices1'FIRST;
      Process:
         loop

            Answer(A_Index) := Vector(V_Index) / Divisor;

            exit Process when V_Index = Indices1'LAST;
            A_Index := Indices2'SUCC(A_Index);
            V_Index := Indices1'SUCC(V_Index);

         end loop Process;

      return Answer;

   end "/";

end Vector_Scalar_Operations_Unconstrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Scalar_Operations_Unconstrained is

pragma PAGE;
   function "*" (Matrix     : Matrices1;
                 Multiplier : Scalars) return Matrices2 is

--   ----------------------
--    --declaration section-
--    -------  --------------

      Answer : Matrices2
                  (Row_Indices2'FIRST ..
                   Row_Indices2'VAL(Matrix'LENGTH(1)-1 +
                                    Row_Indices2'POS(Row_Indices2'FIRST) ),
                   Col_Indices2'FIRST ..
                   Col_Indices2'VAL(Matrix'LENGTH(2)-1 +
                                    Col_Indices2'POS(Col_Indices2'FIRST) ));
      A_Col  : Col_Indices2;
      A_Row  : Row_Indices2;
      M_Col  : Col_Indices1;
      M_Row  : Row_Indices1;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

      A_Row := Row_Indices2'FIRST;
      M_Row := Matrix'FIRST(1);
      Row_Loop:
         loop

            A_Col := Col_Indices2'FIRST;
            M_Col := Matrix'FIRST(2);
            Col_Loop:
               loop

                  Answer(A_Row, A_Col) := Matrix(M_Row, M_Col) * Multiplier;

                  exit Col_Loop when M_Col = Matrix'LAST(2);
                  A_Col := Col_Indices2'SUCC(A_Col);
                  M_Col := Col_Indices1'SUCC(M_Col);

               end loop Col_Loop;

            exit Row_Loop when M_Row = Matrix'LAST(1);
            A_Row := Row_Indices2'SUCC(A_Row);
            M_Row := Row_Indices1'SUCC(M_Row);

         end loop Row_Loop;

      return Answer;

   end "*";
```

```
pragma PAGE;
   function "/" (Matrix  : Matrices2;
                 Divisor : Scalars) return Matrices1 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices1
                 (Row_Indices1'FIRST ..
                  Row_Indices1'VAL(Matrix'LENGTH(1)-1 +
                                   Row_Indices1'POS(Row_Indices1'FIRST) ),
                  Col_Indices1'FIRST ..
                  Col_Indices1'VAL(Matrix'LENGTH(2)-1 +
                                   Col_Indices1'POS(Col_Indices1'FIRST) ));
      A_Col  : Col_Indices1;
      A_Row  : Row_Indices1;
      M_Col  : Col_Indices2;
      M_Row  : Row_Indices2;

-- --------------------
-- --begin function "/"
-- --------------------

   begin

      A_Row := Row_Indices1'FIRST;
      M_Row := Matrix'FIRST(1);
      Row_Loop:
         loop

            A_Col := Col_Indices1'FIRST;
            M_Col := Matrix'FIRST(2);
            Col_Loop:
               loop

                  Answer(A_Row, A_Col) := Matrix(M_Row, M_Col) / Divisor;

                  exit Col_Loop when M_Col = Matrix'LAST(2);
                  A_Col := Col_Indices1'SUCC(A_Col);
                  M_Col := Col_Indices2'SUCC(M_Col);

               end loop Col_Loop;

            exit Row_Loop when M_Row = Matrix'LAST(1);
            A_Row := Row_Indices1'SUCC(A_Row);
            M_Row := Row_Indices2'SUCC(M_Row);

         end loop Row_Loop;

      return Answer;

   end "/";

end Matrix_Scalar_Operations_Unconstrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Diagonal_Matrix_Scalar_Operations is

pragma PAGE;
   function "*" (Matrix     : Diagonal_Matrices1;
                 Multiplier : Scalars) return Diagonal_Matrices2 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Diagonal_Matrices2;
      Index1 : Diagonal_Range1;
      Index2 : Diagonal_Range2;

-- ---------------------
-- --begin function "*"-
-- ---------------------

   begin

      Index1 := Diagonal_Range1'FIRST;
      Index2 := Diagonal_Range2'FIRST;
      Process:
         loop

            Answer(Index2) := Matrix(Index1) * Multiplier;

            exit Process when Index1 = Diagonal_Range1'LAST;
            Index1 := Diagonal_Range1'SUCC(Index1);
            Index2 := Diagonal_Range2'SUCC(Index2);

         end loop Process;

      return Answer;

   end "*";

pragma PAGE;
   function "/" (Matrix  : Diagonal_Matrices2;
                 Divisor : Scalars) return Diagonal_Matrices1 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Diagonal_Matrices1;
      Index1 : Diagonal_Range1;
      Index2 : Diagonal_Range2;

-- ---------------------
-- --begin function "/"-
-- ---------------------

   begin

      Index1 := Diagonal_Range1'FIRST;
```

```
      Index2 := Diagonal_Range2'FIRST;
      Process:
         loop

            Answer(Index1) := Matrix(Index2) / Divisor;

            exit Process when Index1 = Diagonal_Range1'LAST;
            Index1 := Diagonal_Range1'SUCC(Index1);
            Index2 := Diagonal_Range2'SUCC(Index2);

         end loop Process;

      return Answer;

   end "/";

pragma PAGE;
-----------------------------------
--begin processing for package body
-----------------------------------

begin

   --make sure instantiated diagonal matrices are of the same size
   if Diagonal_Matrices1'LENGTH /= Diagonal_Matrices2'LENGTH then
      raise Dimension_Error;
   end if;

end Diagonal_Matrix_Scalar_Operations;
```

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Matrix_Multiply_Unrestricted is

pragma PAGE;
   function "*" (Left  : Left_Matrices;
                 Right : Right_Matrices) return Output_Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer      : Output_Matrices;
      M_Answer    : Output_Row_Indices;
      M_Left      : Left_Row_Indices;
      N_Left      : Left_Col_Indices;
      N_Right     : Right_Row_Indices;
      P_Answer    : Output_Col_Indices;
      P_Right     : Right_Col_Indices;

-- ----------------------
-- --begin of function "*"
-- ----------------------

   begin

      M_Answer := Output_Row_Indices'FIRST;
      M_Left   := Left_Row_Indices'FIRST;
      M_Loop:
         loop

            P_Answer := Output_Col_Indices'FIRST;
            P_Right  := Right_Col_Indices'FIRST;
            P_Loop:
               loop

                  Answer(M_Answer, P_Answer) := 0.0;
                  N_Left                     := Left_Col_Indices'FIRST;
                  N_Right                    := Right_Row_Indices'FIRST;
                  N_Loop:
                     loop

                        Answer(M_Answer, P_Answer) :=
                           Answer(M_Answer, P_Answer) +
                           Left(M_Left, N_Left) * Right(N_Right, P_Right);

                        exit N_Loop when N_Left = Left_Col_Indices'LAST;
                        N_Left  := Left_Col_Indices'SUCC(N_Left);
                        N_Right := Right_Row_Indices'SUCC(N_Right);

                     end loop N_Loop;

                  exit P_Loop when P_Right = Right_Col_Indices'LAST;
                  P_Right  := Right_Col_Indices'SUCC(P_Right);
                  P_Answer := Output_Col_Indices'SUCC(P_Answer);

               end loop P_Loop;
```

```
                exit M_Loop when M_Left = Left_Row_Indices'LAST;
                M_Left   := Left_Row_Indices'SUCC(M_Left);
                M_Answer := Output_Row_Indices'SUCC(M_Answer);

            end loop M_Loop;

        return Answer;

    end "*";

pragma PAGE;
-----------------------------------
--begin processing for package body
-----------------------------------

begin

-- --make sure dimensions are compatible; to be compatible the following
-- --conditions must exist:
-- --must be trying to multiply: [m x n] x [n x p] := [m x p]
    if not (Left_Matrices'LENGTH(2)  = Right_Matrices'LENGTH(1) and      --"n's"
            Left_Matrices'LENGTH(1)  = Output_Matrices'LENGTH(1) and     --"m's"
            Right_Matrices'LENGTH(2) = Output_Matrices'LENGTH(2)) then   --"p's"

--      --dimensions are incompatible
        raise Dimension_Error;

    end if;

end Matrix_Matrix_Multiply_Unrestricted;
```

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Vector_Multiply_Unrestricted is

pragma PAGE;
   function "*" (Matrix : Input_Matrices;
                 Vector : Input_Vectors) return Output_Vectors is

--     ----------------------
--     --declaration section-
--     ----------------------

       Answer   : Output_Vectors;
       M_Answer : Output_Vector_Indices;
       M_Matrix : Row_Indices;
       N_Matrix : Col_Indices;
       N_Vector : Input_Vector_Indices;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

      M_Answer := Output_Vector_Indices'FIRST;
      M_Matrix := Row_Indices'FIRST;
      M_Loop:
         loop

            Answer(M_Answer) := 0.0;
            N_Matrix  := Col_Indices'FIRST;
            N_Vector  := Input_Vector_Indices'FIRST;
            N_Loop:
               loop

                  Answer(M_Answer) := Answer(M_Answer) +
                              Matrix(M_Matrix, N_Matrix) * Vector(N_Vector);

                  exit N_Loop when N_Matrix = Col_Indices'LAST;
                  N_Matrix := Col_Indices'SUCC(N_Matrix);
                  N_Vector := Input_Vector_Indices'SUCC(N_Vector);

               end loop N_Loop;

               exit M_Loop when M_Matrix = Row_Indices'LAST;
               M_Matrix := Row_Indices'SUCC(M_Matrix);
               M_Answer := Output_Vector_Indices'SUCC(M_Answer);

         end loop M_Loop;

      return Answer;

   end "*";

pragma PAGE;
-----------------------------------
--begin processing for package body
-----------------------------------
```

```
begin

-- --make sure dimensions are compatible; for dimensions to be compatible the following
-- --conditions must is what should be requested: [m x n] x [n x 1] = [m x 1]
   if not (Input_Matrices'LENGTH(2) = Input_Vectors'LENGTH and        --"n's"
           Input_Matrices'LENGTH(1) = Output_Vectors'LENGTH) then     --"m's"

--       --dimensions are incompatible
         raise Dimension_Error;

   end if;

end Matrix_Vector_Multiply_Unrestricted;
```

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Vector_Transpose_Multiply_Unrestricted is

pragma PAGE;
   function "*" (Left  : Left_Vectors ;
                 Right : Right_Vectors) return Matrices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer   : Matrices;
       M_Answer : Row_Indices;
       M_Left   : Left_Vector_Indices;
       N_Answer : Col_Indices;
       N_Right  : Right_Vector_Indices;

-- --------------------
-- --begin function "*"
------------------------

   begin

       M_Answer := Row_Indices'FIRST;
       M_Left   := Left_Vector_Indices'FIRST;
       M_Loop:
          loop

               N_Right  := Right_Vector_Indices'FIRST;
               N_Answer := Col_Indices'FIRST;
               N_Loop:
                  loop

                       Answer(M_Answer, N_Answer) := Left(M_Left) * Right(N_Right);

                       exit N_Loop when N_Right = Right_Vector_Indices'LAST;
                       N_Right  := Right_Vector_Indices'SUCC(N_Right);
                       N_Answer := Col_Indices'SUCC(N_Answer);

                  end loop N_Loop;

               exit M_Loop when M_Answer = Row_Indices'LAST;
               M_Answer := Row_Indices'SUCC(M_Answer);
               M_Left   := Left_Vector_Indices'SUCC(M_Left);

       end loop M_Loop;

       return Answer;

   end "*";

pragma PAGE;
-----------------------------------
--begin processing for package body
-----------------------------------

begin
```

```
-- --make sure dimensions are compatible;must have the following conditions:
-- --attempted operation is [m x l] x [l x n] := [m x n]
   if not (Left_Vectors'LENGTH  = Matrices'LENGTH(1) and          --"m's"
           Right_Vectors'LENGTH = Matrices'LENGTH(2)) then        --"n's"

      raise Dimension_Error;

   end if;

end Vector_Vector_Transpose_Multiply_Unrestricted;
```

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Matrix_Transpose_Multiply_Unrestricted is

pragma PAGE;
   function "*" (Left  : Left_Matrices;
                 Right : Right_Matrices) return Output_Matrices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer   : Output_Matrices;
       M_Answer : Output_Row_Indices;
       M_Left   : Left_Row_Indices;
       N_Left   : Left_Col_Indices;
       N_Right  : Right_Col_Indices;
       P_Answer : Output_Col_Indices;
       P_Right  : Right_Row_Indices;

-- --------------------
-- ---begin function "*"
-----------------------

   begin

      M_Answer := Output_Row_Indices'FIRST;
      M_Left   := Left_Row_Indices'FIRST;
      M_Loop:
         loop

            P_Answer := Output_Col_Indices'FIRST;
            P_Right  := Right_Row_Indices'FIRST;
            P_Loop:
               loop

                  Answer(M_Answer, P_Answer) := 0.0;

                  N_Left  := Left_Col_Indices'FIRST;
                  N_Right := Right_Col_Indices'FIRST;
                  N_Loop:
                     loop

                        Answer(M_Answer, P_Answer) :=
                           Answer(M_Answer, P_Answer) +
                           Left(M_Left, N_Left) * Right(P_Right, N_Right);

                        exit N_Loop when N_Left = Left_Col_Indices'LAST;
                        N_Left  := Left_Col_Indices'SUCC(N_Left);
                        N_Right := Right_Col_Indices'SUCC(N_Right);

                     end loop N_Loop;

                  exit P_Loop when P_Answer = Output_Col_Indices'LAST;
                  P_Answer := Output_Col_Indices'SUCC(P_Answer);
                  P_Right  := Right_Row_Indices'SUCC(P_Right);

               end loop P_Loop;
```

```
            exit M_Loop when M_Answer = Output_Row_Indices'LAST;
            M_Answer := Output_Row_Indices'SUCC(M_Answer);
            M_Left   := Left_Row_Indices'SUCC(M_Left);

         end loop M_Loop;

      return Answer;

   end "*";

pragma PAGE;
-----------------------------------
-- begin processing for package body
-----------------------------------
begin

-- --make sure dimension are compatible
-- --need to have: [m x n] x [p x n] := [m x p]
   if not (Left_Matrices'LENGTH(1)  = Output_Matrices'LENGTH(1) and    --"m's"
           Left_Matrices'LENGTH(2)  = Output_Matrices'LENGTH(2) and    --"n's"
           Right_Matrices'LENGTH(1) = Output_Matrices'LENGTH(2)) then  --"p's"

      raise Dimension_Error;

   end if;

end Matrix_Matrix_Transpose_Multiply_Unrestricted;
```

```
separate (General_Vector_Matrix_Algebra)
package body Dot_Product_Operations_Unrestricted is

pragma PAGE;
   function Dot_Product (Left  : Left_Vectors;
                         Right : Right_Vectors) return Result_Elements is

--     ----------------------
--     --declaration section-
--     ----------------------

       Answer  : Result_Elements;
       L_Index : Left_Indices;
       R_Index : Right_Indices;

-- ----------------------------
-- --begin function Dot_Product-
-- ----------------------------

   begin

      Answer  := 0.0;

      L_Index := Left_Indices'FIRST;
      R_Index := Right_Indices'FIRST;
      Process:
         loop

            Answer := Answer + Left(L_Index) * Right(R_Index);

            exit Process when L_Index = Left_Indices'LAST;
            L_Index := Left_Indices'SUCC(L_Index);
            R_Index := Right_Indices'SUCC(R_Index);

         end loop Process;

      return Answer;

   end Dot_Product;

pragma PAGE;
-----------------------------------
--begin processing for package body
-----------------------------------

begin

-- --make sure instantiated vectors are of the same length
   if Left_Vectors'LENGTH /= Right_Vectors'LENGTH then
      raise Dimension_Error;
   end if;

end Dot_Product_Operations_Unrestricted;
```

```
separate (General_Vector_Matrix_Algebra)
package body Diagonal_Full_Matrix_Add_Unrestricted is

pragma PAGE;
   function "+" (D_Matrix : Diagonal_Matrices;
                 F_Matrix : Full_Input_Matrices) return Full_Output_Matrices is

--     ----------------------
--     --declaration section-
--     ----------------------

       Answer       : Full_Output_Matrices;
       A_Col_Index  : Full_Output_Col_Indices;
       A_Col_Marker : Full_Output_Col_Indices;
       A_Row_Index  : Full_Output_Row_Indices;
       D_Index      : Diagonal_Range;
       F_Col_Index  : Full_Input_Col_Indices;
       F_Row_Index  : Full_Input_Row_Indices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

--     --first assign a row full of values, then add in diagonal element

       A_Col_Marker := Full_Output_Col_Indices'FIRST;
       A_Row_Index  := Full_Output_Row_Indices'FIRST;
       D_Index      := Diagonal_Range'FIRST;
       F_Row_Index  := Full_Input_Row_Indices'FIRST;
       Add_Loop:
          loop

             A_Col_Index := Full_Output_Col_Indices'FIRST;
             F_Col_Index := Full_Input_Col_Indices'FIRST;
             Assign_Loop:
                loop

                   Answer(A_Row_Index, A_Col_Index) :=
                      F_Matrix(F_Row_Index, F_Col_Index);

                   exit Assign_Loop
                      when A_Col_Index = Full_Output_Col_Indices'LAST;
                   A_Col_Index := Full_Output_Col_Indices'SUCC(A_Col_Index);
                   F_Col_Index := Full_Input_Col_Indices'SUCC(F_Col_Index);

                end loop Assign_Loop;

             Answer(A_Row_Index, A_Col_Marker) :=
                Answer(A_Row_Index, A_Col_Marker) + D_Matrix(D_Index);

             exit Add_Loop when D_Index = Diagonal_Range'LAST;
             A_Col_Marker := Full_Output_Col_Indices'SUCC(A_Col_Marker);
             A_Row_Index  := Full_Output_Row_Indices'SUCC(A_Row_Index);
             D_Index      := D_Index + 1;
             F_Row_Index  := Full_Input_Row_Indices'SUCC(F_Row_Index);
```

```
         end loop Add_Loop;

      return Answer;

   end "+";

pragma PAGE;
--------------------------------
--begin package body processing
--------------------------------
begin

-- --make sure square matrices of the same size have been instantiated
   if not (Diagonal_Matrices'LENGTH       = Full_Input_Matrices'LENGTH(1)  and
           Full_Input_Matrices'LENGTH(1)  = Full_Input_Matrices'LENGTH(2)  and
           Full_Input_Matrices'LENGTH(1)  = Full_Output_Matrices'LENGTH(1) and
           Full_Output_Matrices'LENGTH(1) = Full_Output_Matrices'LENGTH(2)) then

     raise Dimension_Error;

   end if;

end Diagonal_Full_Matrix_Add_Unrestricted;
```

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Operations_Constrained is

pragma PAGE;
   function "+" (Left  : Vectors;
                 Right : Vectors) return Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer  : Vectors;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Process:
         for INDEX in Indices loop

            Answer(INDEX) := Left(INDEX) + Right(INDEX);

         end loop Process;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Left  : Vectors;
                 Right : Vectors) return Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Vectors;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Process:
         for INDEX in Indices loop

            Answer(INDEX) := Left(INDEX) - Right(INDEX);

         end loop Process;

      return Answer;

   end "-";
```

```
pragma PAGE;
   function Vector_Length (Input : Vectors) return Vector_Elements is

--    ----------------------
--    --declaration section-
--    ----------------------

      Temp   : Vector_Elements_Squared;

-- ------------------------------
-- --begin function Vector_Length
-- ------------------------------

   begin

      Temp := 0.0;

      Process:
         for INDEX in Indices loop
            Temp := Temp +
                    Input(INDEX) * Input(INDEX);
         end loop Process;

      return Sqrt(Temp);

   end Vector_Length;

pragma PAGE;
   function Dot_Product (Left  : Vectors;
                         Right : Vectors) return Vector_Elements_Squared is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Vector_Elements_Squared;

-- ----------------------------
-- --begin function Dot_Product
-- ----------------------------

   begin

      Answer := 0.0;

      Process:
         for INDEX in Indices loop

            Answer := Answer + Left(INDEX) * Right(INDEX);

         end loop Process;

      return Answer;

   end Dot_Product;

end Vector_Operations_Constrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Operations_Constrained is

pragma PAGE;
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for COL in Col_Indices loop

                  Answer(Row, COL) := Left(Row, COL) +
                                      Right(Row, COL);

               end loop Col_Loop;

         end loop Row_Loop;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer   : Matrices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for COL in Col_Indices loop
```

```
                Answer(Row, COL) := Left(Row, COL) -
                                    Right(Row, COL);

            end loop Col_Loop;

         end loop Row_Loop;

      return Answer;

   end "-";

pragma PAGE;
   function "+" (Matrix : Matrices;
                 Addend : Elements) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop
            Col_Loop:
               for COL in Col_Indices loop
                  Answer(Row, COL) := Matrix(Row, COL) + Addend;
               end loop Col_Loop;
         end loop Row_Loop;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Matrix     : Matrices;
                 Subtrahend : Elements) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Row_Loop:
```

```
         for Row in Row_Indices loop
            Col_Loop:
               for COL in Col_Indices loop
                  Answer(Row, COL) := Matrix(Row, COL) - Subtrahend;
               end loop Col_Loop;
         end loop Row_Loop;

      return Answer;

   end "-";

pragma PAGE;
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--    ---------------------
--    --declaration section
--    ---------------------

      COL : Col_Indices;
      Row : Row_Indices;

-- ----------------------------------------
-- --begin procedure Set_To_Identity_Matrix
-- ----------------------------------------

   begin

--    --make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

         Matrix := (others => (others => 0.0));

         COL := Col_Indices'FIRST;
         Row := Row_Indices'FIRST;
         Row_Loop:
            loop

--             --set diagonal element equal to 1
               Matrix(Row, COL) := 1.0;

               exit when Row = Row_Indices'LAST;
               COL := Col_Indices'SUCC(COL);
               Row := Row_Indices'SUCC(Row);

            end loop Row_Loop;

      else

--          --do not have a square matrix
         raise Dimension_Error;

      end if;

   end Set_To_Identity_Matrix;

pragma PAGE;
   procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
```

```
   begin
      Matrix := (others => (others => 0.0));
   end Set_To_Zero_Matrix;
end Matrix_Operations_Constrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Dynamically_Sparse_Matrix_Operations_Constrained is

pragma PAGE;
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--     ---------------------
--     --declaration section
--     ---------------------

       COL : Col_Indices;
       Row : Row_Indices;

-- ----------------------------------------
-- --begin procedure Set_to_Identity_Matrix
-- ----------------------------------------

   begin

--     --make sure input matrix is a square matrix
       if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

          Matrix := (others => (others => 0.0));

          COL := Col_Indices'FIRST;
          Row := Row_Indices'FIRST;
          Row_Loop:
             loop

--              --set diagonal element equal to 1.0
                Matrix(Row, COL) := 1.0;

                exit when Row = Row_Indices'LAST;
                COL := Col_Indices'SUCC(COL);
                Row := Row_Indices'SUCC(Row);

             end loop Row_Loop;

       else

          raise Dimension_Error;

       end if;

   end Set_To_Identity_Matrix;

pragma PAGE;
   procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

   begin

      Matrix := (others => (others => 0.0));

   end Set_To_Zero_Matrix;

pragma PAGE;
   function Add_To_Identity (Input : Matrices) return Matrices is
```

```
--     ---------------------
--     --declaration section
--     ---------------------

       Answer : Matrices;
       COL    : Col_Indices;
       Row    : Row_Indices;

-- --------------------------------
-- --begin function Add_to_Identity
-- --------------------------------

   begin

--     --make sure input is a square matrix
       if Input'LENGTH(1) = Input'LENGTH(2) then

          Answer := Input;

--        --add "identity" values to diagonal elements
          COL := Col_Indices'FIRST;
          Row := Row_Indices'FIRST;
          Row_Loop:
             loop

                if Answer(Row, COL) /= 0.0 then
                   Answer(Row, COL) := Answer(Row, COL) + 1.0;
                else
                   Answer(Row, COL) := 1.0;
                end if;

                exit when Row = Row_Indices'LAST;
                COL := Col_Indices'SUCC(COL);
                Row := Row_Indices'SUCC(Row);

             end loop Row_Loop;

       else

          raise Dimension_Error;

       end if;

       return Answer;

   end Add_To_Identity;

pragma PAGE;
   function Subtract_From_Identity (Input : Matrices) return Matrices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer  : Matrices;
       COL     : Col_Indices;
```

```
      Row     : Row_Indices;

-- ------------------------------------
-- --begin procedure Subtract_From_Identity
-- ------------------------------------

   begin

--     --make sure input is a square matrix
       if Input'LENGTH(1) = Input'LENGTH(2) then

          COL := Col_Indices'FIRST;
          Row := Row_Indices'FIRST;
          Row_Loop:
             loop

                Col_Loop:
                   for Temp_Col in Col_Indices loop
                      if Input(Row,Temp_Col) /= 0.0 then
                         Answer(Row,Temp_Col) := - Input(Row,Temp_Col);
                      else
                         Answer(Row,Temp_Col) := 0.0;
                      end if;
                   end loop Col_Loop;

                if Answer(Row, COL) /= 0.0 then
                   Answer(Row, COL) := Answer(Row, COL) + 1.0;
                else
                   Answer(Row, COL) := 1.0;
                end if;

                exit when Row = Row_Indices'LAST;
                COL := Col_Indices'SUCC(COL);
                Row := Row_Indices'SUCC(Row);

             end loop Row_Loop;

       else

          raise Dimension_Error;

       end if;

       return Answer;

   end Subtract_From_Identity;

pragma PAGE;
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--     ---------------------
--     --declaration section
--     ---------------------

       Answer : Matrices;
```

```
-- --------------------
-- --begin function "+"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for COL in Col_Indices loop

                  if Left(Row, COL) = 0.0 then
                     if Right(Row, COL) = 0.0 then
                        Answer(Row, COL) := 0.0;
                     else
                        Answer(Row, COL) := Right(Row, COL);
                     end if;
                  elsif Right(Row, COL) = 0.0 then
                     Answer(Row, COL) := Left(Row, COL);
                  else
                     Answer(Row, COL) := Left(Row, COL) +
                                         Right(Row, COL);
                  end if;

               end loop Col_Loop;

         end loop Row_Loop;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer : Matrices;

-- --------------------
-- --begin function "-"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for COL in Col_Indices loop

                  if Left(Row, COL) = 0.0 then
                     if Right(Row, COL) = 0.0 then
```

```
                        Answer(Row, COL) := 0.0;
                     else
                        Answer(Row, COL) := - Right(Row, COL);
                     end if;
                  elsif Right(Row, COL) = 0.0 then
                     Answer(Row, COL) := Left(Row, COL);
                  else
                     Answer(Row, COL) := Left(Row, COL) -
                                         Right(Row, COL);
                  end if;

               end loop Col_Loop;

            end loop Row_Loop;

         return Answer;

      end "-";

end Dynamically_Sparse_Matrix_Operations_Constrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Symmetric_Full_Storage_Matrix_Operations_Constrained is

pragma PAGE;
   procedure Change_Element (New_Value : in     Elements;
                             Row       : in     Row_Indices;
                             COL       : in     Col_Indices;
                             Matrix    : in out Matrices) is

--    ----------------------
--    --declaration section-
--    ----------------------

      S_Col : Col_Indices;
      S_Row : Row_Indices;

-- ---------------------------------
-- --begin procedure Change_Element-
-- ---------------------------------

   begin

      S_Col := Col_Indices'VAL(Row_Indices'POS(Row) -
                               Row_Indices'POS(Row_Indices'FIRST) +
                               Col_Indices'POS(Col_Indices'FIRST));

      S_Row := Row_Indices'VAL(Col_Indices'POS(COL) -
                               Col_Indices'POS(Col_Indices'FIRST) +
                               Row_Indices'POS(Row_Indices'FIRST));

      Matrix(Row, COL) := New_Value;
      Matrix(S_Row, S_Col) := New_Value;

   end Change_Element;

pragma PAGE;
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

--    ---------------------
--    --declaration section
--    ---------------------

      COL : Col_Indices;
      Row : Row_Indices;

-- ----------------------------------------
-- --begin procedure Set_to_Identity_Matrix
-- ----------------------------------------

   begin

      Matrix := (others => (others => 0.0));

      COL := Col_Indices'FIRST;
      Row := Row_Indices'FIRST;
      Row_Loop:
         loop
```

```
--             --set diagonal element equal to
               Matrix(Row, COL) := 1.0;

               exit when Row = Row_Indices'LAST;
               COL := Col_Indices'SUCC(COL);
               Row := Row_Indices'SUCC(Row);

            end loop Row_Loop;

      end Set_To_Identity_Matrix;

pragma PAGE;
      procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

      begin

         Matrix := (others => (others => 0.0));

      end Set_To_Zero_Matrix;

pragma PAGE;
      function Add_To_Identity (Input : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

         Answer      : Matrices;
         COL         : Col_Indices;
         Row         : Row_Indices;

-- --------------------------------
-- --begin function Add_to_Identity
-- --------------------------------

      begin

         Answer := Input;

         COL := Col_Indices'FIRST;
         Row := Row_Indices'FIRST;
         Access_Diagonal_Elements:
            loop

               Answer(Row,COL) := Answer(Row,COL) + 1.0;

               exit when Row = Row_Indices'LAST;
               COL := Col_Indices'SUCC(COL);
               Row := Row_Indices'SUCC(Row);

            end loop Access_Diagonal_Elements;

         return Answer;

      end Add_To_Identity;
```

```
pragma PAGE;
   function Subtract_From_Identity (Input : Matrices) return Matrices is

--     ----------------------
--     --declaration section
--     ----------------------

       Answer     : Matrices;
       Row        : Row_Indices;
       S_Col      : Col_Indices;
       S_Row      : Row_Indices;

-- ---------------------------------------
-- --begin function Subtract_from_Identity
-- ---------------------------------------

   begin

--     --handle first diagonal element

       Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
          1.0 - Input(Row_Indices'FIRST, Col_Indices'FIRST);

--     --will subtract the remaining of the input matrix from an identity matrix
--     --by doing the following:
--     --   o subtracting the nondiagonal elements in the bottom half of the
--     --        matrix from 0.0,
--     --   o assigning values obtained in the bottom half of the matrix to the
--     --        symmetric elements in the top half of the matrix, and then
--     --   o subtracting the diagonal elements from 1.0

--     --S_Col will go across the columns as Row goes down the rows to keep
--     --track of the column containing the diagonal element
       S_Col := Col_Indices'SUCC(Col_Indices'FIRST);
       Row   := Row_Indices'SUCC(Row_Indices'FIRST);
       Do_Every_Row_Except_First:
          loop

--           --S_Row will go down the rows as Col goes across the columns
             S_Row := Row_Indices'FIRST;
             Subtract_Nondiagonal_Elements_From_Zero:
                for COL in Col_Indices'FIRST ..
                           Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop

                   Answer(Row,COL) := - Input(Row,COL);

                   Answer(S_Row,S_Col) := Answer(Row,COL);

                   S_Row     := Row_Indices'SUCC(S_Row);

                end loop Subtract_Nondiagonal_Elements_From_Zero;

--           --subtract diagonal element from 1.0
             Answer(Row, S_Col) := 1.0 - Input(Row, S_Col);

             exit when Row = Row_Indices'LAST;
             S_Col := Col_Indices'SUCC(S_Col);
```

```
            Row   := Row_Indices'SUCC(Row);

         end loop Do_Every_Row_Except_First;

      return Answer;

   end Subtract_From_Identity;

pragma PAGE;
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ---------------------
--    --declaration section
--    ---------------------

      Answer    : Matrices;
      Row       : Row_Indices;
      S_Col     : Col_Indices;
      S_Row     : Row_Indices;

-- ---------------------
-- --begin function "+"
-- ---------------------

   begin

--    --handle first diagonal element
      Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
         Left(Row_Indices'FIRST, Col_Indices'FIRST) +
         Right(Row_Indices'FIRST, Col_Indices'FIRST);

--    --addition calculations will only be carried out on the bottom half
--    --of the input matrices followed by assignments to the symmetric
--    --elements in the top half of the matrix

--    --as Row goes down the rows, S_Col will go across the columns to keep
--    --track of the column containing the diagonal element
      S_Col := Col_Indices'SUCC(Col_Indices'FIRST);
      Row := Row_Indices'SUCC(Row_Indices'FIRST);
      Do_All_Rows_Except_First:
         loop

--          --as Col goes across the columns, S_Row will go down the rows;
            S_Row := Row_Indices'FIRST;
            Add_Bottom_Half_Elements:
               for COL in Col_Indices'FIRST ..
                          Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop

--                --add elements in bottom half of the matrix
                  Answer(Row,COL) := Left(Row, COL) + Right(Row, COL);

--                --assign value to symmetric element in top half of matrix
                  Answer(S_Row,S_Col) := Answer(Row, COL);

                  S_Row     := Row_Indices'SUCC(S_Row);
```

```
                end loop Add_Bottom_Half_Elements;

--              --add diagonal elements together
                Answer(Row, S_Col) := Left(Row,S_Col) + Right(Row,S_Col);

                exit when Row = Row_Indices'LAST;
                S_Col := Col_Indices'SUCC(S_Col);
                Row := Row_Indices'SUCC(Row);

            end loop Do_All_Rows_Except_First;

        return Answer;

    end "+";

pragma PAGE;
    function "-" (Left  : Matrices;
                  Right : Matrices) return Matrices is

--      ---------------------
--      --declaration section
--      ---------------------

        Answer    : Matrices;
        Row       : Row_Indices;
        S_Col     : Col_Indices;
        S_Row     : Row_Indices;

-- ---------------------
-- --begin function "-"
-- ---------------------

    begin

--      --handle first diagonal element
        Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
           Left(Row_Indices'FIRST, Col_Indices'FIRST) -
           Right(Row_Indices'FIRST, Col_Indices'FIRST);

--      --subtraction calculations will only be carried out on the bottom half
--      --of the input matrices followed by assignments to the symmetric
--      --elements in the top half of the matrix

--      --as Row goes down the rows, S_Col will go across the columns to keep
--      --track of the column containing the diagonal element
        S_Col := Col_Indices'SUCC(Col_Indices'FIRST);
        Row := Row_Indices'SUCC(Row_Indices'FIRST);
        Do_All_Rows_Except_First:
           loop

--              --as Col goes across the columns, S_Row will go down the rows;
                S_Row := Row_Indices'FIRST;
                Subtract_Bottom_Half_Elements:
                   for COL in Col_Indices'FIRST ..
                              Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop

--                      --subtract elements in bottom half of the matrix
```

```
                    Answer(Row,COL) := Left(Row, COL) - Right(Row, COL);

--                  --assign value to symmetric element in top half of matrix
                    Answer(S_Row,S_Col) := Answer(Row, COL);

                    S_Row      := Row_Indices'SUCC(S_Row);

                 end loop Subtract_Bottom_Half_Elements;

--               --subtract diagonal elements together
                 Answer(Row, S_Col) := Left(Row,S_Col) - Right(Row,S_Col);

                 exit when Row = Row_Indices'LAST;
                 S_Col := Col_Indices'SUCC(S_Col);
                 Row := Row_Indices'SUCC(Row);

              end loop Do_All_Rows_Except_First;

           return Answer;

        end "-";

pragma PAGE;
-------------------------------------------
--processing for Symmetric_Full_Storage_
--Matrix_Operations_Constrained package body
-------------------------------------------

begin

   if Matrices'LENGTH(1) /= Matrices'LENGTH(2) then
      raise Dimension_Error;
   end if;

end Symmetric_Full_Storage_Matrix_Operations_Constrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Scalar_Operations_Constrained is

pragma PAGE;
   function "*" (Vector     : Vectors2;
                 Multiplier : Scalars) return Vectors1 is

--     ----------------------
--     --declaration section-
--     ----------------------

       Answer  : Vectors1;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

     Process:
        for INDEX in Indices loop

           Answer(INDEX) := Vector(INDEX) * Multiplier;

        end loop Process;

     return Answer;

   end "*";


pragma PAGE;
   function "/" (Vector  : Vectors1;
                 Divisor : Scalars) return Vectors2 is

--     ----------------------
--     --declaration section-
--     ----------------------

       Answer  : Vectors2;

-- -----------------------------------
-- --begin function Vector_Scalar_Divide
-- -----------------------------------

   begin

      Process:
         for INDEX in Indices loop

            Answer(INDEX) := Vector(INDEX) / Divisor;

         end loop Process;

      return Answer;

   end "/";
```

```
end Vector_Scalar_Operations_Constrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Scalar_Operations_Constrained is

pragma PAGE;
   function "*" (Matrix     : Matrices1;
                 Multiplier : Scalars) return Matrices2 is

--    ----------------------
--      --declaration section-
--     ----------------------

      Answer : Matrices2;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for COL in Col_Indices loop

                  Answer(Row, COL) := Matrix(Row, COL) * Multiplier;

               end loop Col_Loop;

         end loop Row_Loop;

      return Answer;

   end "*";

pragma PAGE;
   function "/" (Matrix  : Matrices2;
                 Divisor : Scalars) return Matrices1 is

--     ----------------------
--     --declaration section-
--     ----------------------

      Answer : Matrices1;

-- --------------------
-- --begin function "/"
-- --------------------

   begin

      Row_Loop:
         for Row in Row_Indices loop

            Col_Loop:
               for COL in Col_Indices loop
```

```
                    Answer(Row, COL) := Matrix(Row, COL) / Divisor;
                end loop Col_Loop;
            end loop Row_Loop;
        return Answer;
    end "/";
end Matrix_Scalar_Operations_Constrained;
```

```
separate (General_Vector_Matrix_Algebra)
function Matrix_Matrix_Multiply_Restricted (Left  : Left_Matrices;
                Right : Right_Matrices) return Output_Matrices is

-- ----------------------
-- --declaration section-
-- ----------------------

   Answer      : Output_Matrices;

-------------------------------------------------------
--begin of function Matrix_Matrix_Multiply_Restricted
-------------------------------------------------------

begin

   Answer := (others => (others => 0.0));

   M_Loop:
      for M in M_Indices loop

         P_Loop:
            for P in P_Indices loop

               N_Loop:
                  for N in N_Indices loop

                     Answer(M, P) := Answer(M, P) +
                                     Left(M, N) * Right(N, P);

                  end loop N_Loop;

            end loop P_Loop;

      end loop M_Loop;

   return Answer;

end Matrix_Matrix_Multiply_Restricted;
```

```
separate (General_Vector_Matrix_Algebra)
function Matrix_Vector_Multiply_Restricted
             (Matrix : Input_Matrices;
              Vector : Input_Vectors) return Output_Vectors is

-- ----------------------
-- --declaration section-
-- ----------------------

   Answer    : Output_Vectors;

-------------------------------------------------
--begin function Matrix_Vector_Multiply_Restricted
-------------------------------------------------

begin

   Answer := (others => 0.0);

   M_Loop:
      for M in Indices1 loop

         N_Loop:
            for N in Indices2 loop

               Answer(M) := Answer(M) +
                            Matrix(M, N) * Vector(N);

            end loop N_Loop;

      end loop M_Loop;

   return Answer;

end Matrix_Vector_Multiply_Restricted;
```

```
separate (General_Vector_Matrix_Algebra)
function Vector_Vector_Transpose_Multiply_Restricted
            (Left  : Left_Vectors ;
             Right : Right_Vectors) return Matrices is

-- ---------------------
-- --declaration section
-- ---------------------

   Answer   : Matrices;

-------------------------------------------------------------
--begin function Vector_Vector_Transpose_Multiply_Restricted
-------------------------------------------------------------

begin

   M_Loop:
      for M in Indices1 loop

         N_Loop:
            for N in Indices2 loop

               Answer(M, N) := Left(M) * Right(N);

            end loop N_Loop;

   end loop M_Loop;

   return Answer;

end Vector_Vector_Transpose_Multiply_Restricted;
```

```
separate (General_Vector_Matrix_Algebra)
function Matrix_Matrix_Transpose_Multiply_Restricted
            (Left  : Left_Matrices;
            Right : Right_Matrices) return Output_Matrices is

-- ---------------------
-- --declaration section
-- ---------------------

   Answer   : Output_Matrices;

-------------------------------------------------------------
--begin function Matrix_Matrix_Transpose_Multiply_Restricted
-------------------------------------------------------------

begin

   Answer := (others => (others => 0.0));

   M_Loop:
      for M in M_Indices loop

         P_Loop:
            for P in P_Indices loop

               N_Loop:
                  for N in N_Indices loop

                     Answer(M, P) := Answer(M, P) +
                                     Left(M, N) * Right(P, N);

                  end loop N_Loop;

            end loop P_Loop;

      end loop M_Loop;

   return Answer;

end Matrix_Matrix_Transpose_Multiply_Restricted;
```

```
separate (General_Vector_Matrix_Algebra)
function Dot_Product_Operations_Restricted
            (Left  : Left_Vectors;
             Right : Right_Vectors) return Result_Elements is

-- ----------------------
-- --declaration section-
-- ----------------------

   Answer  : Result_Elements;

-------------------------------------------------
--begin function Dot_Product_Operations_Restricted
-------------------------------------------------

begin

   Answer  := 0.0;

   Process:
      for INDEX in Indices loop

         Answer := Answer + Left(INDEX) * Right(INDEX);

      end loop Process;

   return Answer;

end Dot_Product_Operations_Restricted;
```

```
separate (General_Vector_Matrix_Algebra)
function Diagonal_Full_Matrix_Add_Restricted
            (D_Matrix : Diagonal_Matrices;
             F_Matrix : Full_Matrices) return Full_Matrices is

-- ----------------------
-- --declaration section-
-- ----------------------

   Answer       : Full_Matrices;
   Diag_Index   : Diagonal_Range;
   INDEX        : Indices;

-- ------------------------------------------------
-- -begin function Diagonal_Full_Matrix_Add_Restricted
-- ------------------------------------------------

begin

-- --assign all values to answer and then add in diagonal elements
   Answer := F_Matrix;

-- --now add in diagonal elements

   Diag_Index := Diagonal_Range'FIRST;
   INDEX      := Indices'FIRST;
   Add_Loop:
      loop

         Answer(INDEX, INDEX) := Answer(INDEX, INDEX) + D_Matrix(Diag_Index);

         exit when INDEX = Indices'LAST;
         Diag_Index := Diagonal_Range'SUCC(Diag_Index);
         INDEX      := Indices'SUCC(INDEX);

      end loop Add_Loop;

   return Answer;

end Diagonal_Full_Matrix_Add_Restricted;
```

```
separate (General_Vector_Matrix_Algebra)
package body Vector_Matrix_Multiply_Unrestricted is

   function "*" (Vector : Input_Vectors;
                 Matrix : Input_Matrices) return Output_Vectors is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Output_Vectors := (others => 0.0);
      M_V    : Input_Vector_Indices;
      N_A    : Output_Vector_Indices;
      N      : Col_Indices;
      M      : Row_Indices;

-- --------------------
-- --begin function "*"
-- --------------------

   begin

      N_A := Output_Vector_Indices'FIRST;
      N   := Col_Indices'FIRST;
      N_Loop:
         loop

            M_V := Input_Vector_Indices'FIRST;
            M   := Row_Indices'FIRST;
            M_Loop:
               loop

                  Answer (N_A) := Answer(N_A) + Vector(M_V) * Matrix(M, N);

                  exit when M = Row_Indices'LAST;
                  M := Row_Indices'SUCC(M);
                  M_V := Input_Vector_Indices'SUCC(M_V);

               end loop M_Loop;

            exit when N = Col_Indices'LAST;
            N := Col_Indices'SUCC(N);
            N_A := Output_Vector_Indices'SUCC(N_A);

         end loop N_Loop;

      return Answer;

   end "*";

pragma PAGE;
-------------------------------------------------
--begin package Vector_Matrix_Multiply_Unrestricted
-------------------------------------------------

begin
```

```
-- --make sure package has been instantiated with the correct dimensions;
-- --the following dimensions are expected:  [1xm] * [mxn] => [1xn]

   if Input_Vectors'LENGTH /= Input_Matrices'LENGTH(1) or      --m's not equal
      Input_Matrices'LENGTH(2) /= Output_Vectors'LENGTH then   --n's not equal

      raise Dimension_Error;

   end if;

end Vector_Matrix_Multiply_Unrestricted;
```

```
separate (General_Vector_Matrix_Algebra)
function Vector_Matrix_Multiply_Restricted
            (Vector : Input_Vectors;
             Matrix : Input_Matrices) return Output_Vectors is

-- ---------------------
-- --declaration section
-- ---------------------

   Answer : Output_Vectors := (others => 0.0);

-------------------------------------------------
--begin function Vector_Matrix_Multiply_Restricted
-------------------------------------------------

begin

   N_Loop:
      for N in Indices2 loop

         M_Loop:
            for M in Indices1 loop

               Answer(N) := Answer(N) + Vector(M) * Matrix(M,N);

            end loop M_Loop;

      end loop N_Loop;

   return Answer;

end Vector_Matrix_Multiply_Restricted;
```

```
separate (General_Vector_Matrix_Algebra)
package body Aba_Trans_Dynam_Sparse_Matrix_Sq_Matrix is


   function Sparse_Left_Multiply(Left  : A_Elements;
                                 Right : B_Elements ) return A_Elements is
      Answer : A_Elements;
   begin
      if Left = 0.0 then
         Answer := 0.0;
      else
         Answer := Left * A_Elements( Right );
      end if;
      return Answer;
   end Sparse_Left_Multiply;



   function Sparse_Right_Multiply( Left  : A_Elements;
                                   Right : A_Elements ) return C_Elements is
      Answer : C_Elements;
   begin
      if Right = 0.0 then
         Answer := 0.0;
      else
         Answer := C_Elements( Left * Right );
      end if;
      return Answer;
   end Sparse_Right_Multiply;



   function Matrix_Multiply is new Matrix_Matrix_Multiply_Restricted
      ( Left_Elements   => A_Elements,
        Right_Elements  => B_Elements,
        Output_Elements => A_Elements,
        M_Indices       => M_Indices,
        N_Indices       => N_Indices,
        P_Indices       => N_Indices,
        Left_Matrices   => A_Matrices,
        Right_Matrices  => B_Matrices,
        Output_Matrices => A_Matrices,
        "*"             => Sparse_Left_Multiply );



   function Matrix_Transpose_Multiply is new
      Matrix_Matrix_Transpose_Multiply_Restricted
         ( Left_Elements   => A_Elements,
           Right_Elements  => A_Elements,
           Output_Elements => C_Elements,
           M_Indices       => M_Indices,
           N_Indices       => N_Indices,
           P_Indices       => M_Indices,
           Left_Matrices   => A_Matrices,
           Right_Matrices  => A_Matrices,
           Output_Matrices => C_Matrices,
```

```
          "*"              => Sparse_Right_Multiply );

   pragma PAGE;
   function Aba_Transpose( A : A_Matrices;
                           B : B_Matrices )
                                return C_Matrices is

      Intermediate : A_Matrices;
      Answer       : C_Matrices;

   begin

--    ------------------
--    - multiply A * B -
--    ------------------
      Intermediate := Matrix_Multiply( Left  => A,
                                       Right => B );


--    --------------------------------
--    - multiply AB * transpose of A -
--    --------------------------------
      Answer := Matrix_Transpose_Multiply( Left  => Intermediate,
                                           Right => A );
      return Answer;

   end Aba_Transpose;

end Aba_Trans_Dynam_Sparse_Matrix_Sq_Matrix;
```

```
separate (General_Vector_Matrix_Algebra)
package body Aba_Trans_Vector_Sq_Matrix is


   function Multiply_Vm( Left  : Vector_Elements;
                         Right : Matrix_Elements ) return Vector_Elements is
   begin
      return Left * Vector_Elements( Right );
   end Multiply_Vm;



   function Multiply_Vv( Left : Vector_Elements;
                         Right: Vector_Elements ) return Scalars is
   begin
      return Scalars( Left ) * Scalars( Right );
   end Multiply_Vv;



   function Vector_Matrix_Multiply is new Vector_Matrix_Multiply_Restricted
           ( Input_Vector_Elements  => Vector_Elements,
             Matrix_Elements        => Matrix_Elements,
             Output_Vector_Elements => Vector_Elements,
             Indices1               => Indices,
             Indices2               => Indices,
             Input_Vectors          => Vectors,
             Input_Matrices         => Matrices,
             Output_Vectors         => Vectors,
             "*"                    => Multiply_Vm );



   function Vector_Vector_Multiply is new Dot_Product_Operations_Restricted
              ( Left_Elements   => Vector_Elements,
                Right_Elements  => Vector_Elements,
                Result_Elements => Scalars,
                Indices         => Indices,
                Left_Vectors    => Vectors,
                Right_Vectors   => Vectors,
                "*"             => Multiply_Vv );

   pragma PAGE;
   function Aba_Transpose( A : Vectors;
                           B : Matrices ) return Scalars is

      Partial_Answer : Vectors;
      Answer         : Scalars;

   begin

--    ------------------
--    - multiply A * B -
--    ------------------
      Partial_Answer := Vector_Matrix_Multiply( Vector => A,
                                                Matrix => B );
```

```
--      -------------------------------
--      - multiply AB * transpose of A -
--      -------------------------------
        Answer := Vector_Vector_Multiply( Left  => Partial_Answer,
                                          Right => A );

        return Answer;

    end Aba_Transpose;

end Aba_Trans_Vector_Sq_Matrix;
```

```
separate (General_Vector_Matrix_Algebra)
package body Aba_Trans_Vector_Scalar is

-- ------------------------------------------
-- -- Operators provided for instantiations -
------------------------------------------

   function Multiply_Vs( Left  : Vector_Elements;
                         Right : Scalars )  return Vector_Elements is
   begin
      return Left * Vector_Elements( Right );
   end Multiply_Vs;


-- ------------------------------------------------------------------------
-- -- This operator is not used, but is required for the instantiation. -
-- -- It is "dummied" out to make it as small as possible.              -
-- ------------------------------------------------------------------------
   function Divide_Vs( Left  : Vector_Elements;
                       Right : Scalars )  return Vector_Elements is
   begin
      return Left;
   end Divide_Vs;


   function Multiply_Vv( Left  : Vector_Elements;
                         Right : Vector_Elements ) return Matrix_Elements is
   begin
      return Matrix_Elements( Left ) * Matrix_Elements( Right );
   end Multiply_Vv;


-- ------------------------------------
-- -- Instantiations for ABA transpose -
--------------------------------------

   package Vs_Opns is new Vector_Scalar_Operations_Constrained
               ( Elements1 => Vector_Elements,
                 Elements2 => Vector_Elements,
                 Scalars   => Scalars,
                 Indices   => Indices,
                 Vectors1  => Vectors,
                 Vectors2  => Vectors,
                 "*"       => Multiply_Vs,
                 "/"       => Divide_Vs );

   use Vs_Opns;


   function Vv_Transpose_Multiply is new
      Vector_Vector_Transpose_Multiply_Restricted
         ( Left_Vector_Elements  => Vector_Elements,
           Right_Vector_Elements => Vector_Elements,
           Matrix_Elements       => Matrix_Elements,
           Indices1              => Indices,
           Indices2              => Indices,
```

```
            Left_Vectors          => Vectors,
            Right_Vectors         => Vectors,
            Matrices              => Matrices,
            "*"                   => Multiply_Vv );

   pragma PAGE;
   function Aba_Transpose( A : Vectors;
                           B : Scalars ) return Matrices is

      Partial_Answer : Vectors;
      Answer         : Matrices;

   begin

--    ------------------
--    - multiply A * B -
--    ------------------
      Partial_Answer := A * B;

--    --------------------------------
--    - multiply AB * transpose of A -
--    --------------------------------
      Answer := Vv_Transpose_Multiply( Left  => Partial_Answer,
                                       Right => A );
      return Answer;

   end Aba_Transpose;

end Aba_Trans_Vector_Scalar;
```

```
separate (General_Vector_Matrix_Algebra)
package body Column_Matrix_Operations is

pragma PAGE;
   function Set_Diagonal_And_Subtract_From_Identity
      ( Column        : Vectors;
        Active_Column : Indices ) return Column_Matrices is

      Answer : Column_Matrices;

   begin

      Answer.Col_Vector    := Column;
      Answer.Diagonal      := TRUE;
      Answer.Active_Column := Active_Column;
      Range_Loop:
         for INDEX in Indices loop
            Answer.Col_Vector( INDEX ) := - Answer.Col_Vector( INDEX );
         end loop Range_Loop;
      Answer.Col_Vector(Active_Column) := Answer.Col_Vector(Active_Column) +
                                          1.0;
      return Answer;

   end Set_Diagonal_And_Subtract_From_Identity;

pragma PAGE;
   function Aba_Transpose( A : Column_Matrices;
                           B : B_Matrices ) return C_Matrices is

      Answer      : C_Matrices;
      Temp_Vector : Vectors := A.Col_Vector;

   begin
      if A.Diagonal then
         Temp_Vector( A.Active_Column ) := Temp_Vector( A.Active_Column ) - 1.0;
         M_Loop:
            for Row in Indices loop
               P_Loop:
                  for COL in Indices loop
                     Answer( Row, COL ) := C_Matrix_Elements(
                        Temp_Vector( Row ) * Temp_Vector( COL ) *
                        B( A.Active_Column, A.Active_Column )          +
                        Temp_Vector( COL ) * B( A.Active_Column, Row ) +
                        Temp_Vector( Row ) * B( A.Active_Column, COL ) +
                        B( Row, COL ) );
                  end loop P_Loop;
            end loop M_Loop;
      else
         M1_Loop:
            for Row in Indices loop
               P1_Loop:
                  for COL in Indices loop
                     Answer( Row, COL ) := C_Matrix_Elements(
                        A.Col_Vector( Row ) * A.Col_Vector( COL ) *
                        B( A.Active_Column, A.Active_Column ) );
                  end loop P1_Loop;
            end loop M1_Loop;
```

```
      end if;
      return Answer;

   end Aba_Transpose;

pragma PAGE;
   function Aba_Symm_Transpose( A : Column_Matrices;
                                B : B_Matrices ) return C_Matrices is

      Answer      : C_Matrices;
      LAST        : Indices;
      Temp_Vector : Vectors := A.Col_Vector;

   begin

      LAST := Indices'LAST;
      if A.Diagonal then     -- Diagonal value is 1 --
         Temp_Vector( A.Active_Column ) := Temp_Vector( A.Active_Column ) - 1.0;
         M_Loop:
            for Row in Indices loop
               P_Loop:
--                ---------------------
--                - Calculate values -
--                ---------------------
                  for COL in Row .. Indices'LAST loop
                     Answer( Row, COL ) := C_Matrix_Elements(
                        Temp_Vector( Row ) * Temp_Vector( COL ) *
                        B( A.Active_Column, A.Active_Column )          +
                        Temp_Vector( COL ) * B( A.Active_Column, Row ) +
                        Temp_Vector( Row ) * B( A.Active_Column, COL ) +
                        B( Row, COL ) );
--                   ------------------------------------------
--                   - Assign calculated value to corresponding -
--                   - lower triangular position                -
--                   ------------------------------------------
                     Answer( COL, Row ) := Answer( Row, COL );
                  end loop P_Loop;
            end loop M_Loop;
      else                        -- diagonal value is 0 --
         M1_Loop:
            for Row in Indices loop
               P1_Loop:
--                ---------------------
--                - Calculate values -
--                ---------------------
                  for COL in Row .. Indices'LAST loop
                     Answer( Row, COL ) := C_Matrix_Elements(
                        A.Col_Vector( Row ) * A.Col_Vector( COL ) *
                        B( A.Active_Column, A.Active_Column ) );
--                   ------------------------------------------
--                   - Assign calculated value to corresponding -
--                   - lower triangular position                -
--                   ------------------------------------------
                     Answer( COL, Row ) := Answer( Row, COL );
                  end loop P1_Loop;
            end loop M1_Loop;
      end if;                     -- Diagonal value is 0 --
```

```
      return Answer;
   end Aba_Symm_Transpose;
end Column_Matrix_Operations;
```

(This page left intentionally blank.)

SUPPLEMENTARY

INFORMATION

DEPARTMENT OF THE AIR FORCE
WRIGHT LABORATORY (AFSC)
EGLIN AIR FORCE BASE, FLORIDA, 32542-5434

ERRATA
AD-B120 252

REPLY TO ATTN OF: MNOI

13 Feb 92

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center
ATTN: DTIC/HAR (Mr William Bush)
Bldg 5, Cameron Station
Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

	Technical Report Number	AD Number
1.	88-18-Vol-4	ADB 120 251
2.	88-18-Vol-5	ADB 120 252
3.	88-18-Vol-6	ADB 120 253
4.	88-25-Vol-1	ADB 120 309
5.	88-25-Vol-2	ADB 120 310
6.	88-62-Vol-1	ADB 129 568
7.	88-62-Vol-2	ADB 129 569
8.	88-62-Vol-3	ADB 129-570
9.	85-93-Vol-1	ADB 102-654
10.	85-93-Vol-2	ADB 102-655
11.	85-93-Vol-3	ADB 102-656
12.	88-18-Vol-1	ADB 120 248
13.	88-18-Vol-2	ADB 120 249
14.	88-18-Vol-7	ADB 120 254
15.	88-18-Vol-8	ADB 120 255
16.	88-18-Vol-9	ADB 120 256
17.	88-18-Vol-10	ADB 120 257
18.	88-18-Vol-11	ADB 120 258
19.	88-18-Vol-12	ADB 120 259

2. If you have any questions regarding this request call me at DSN 872-4620.

LYNN S. WARGO
Chief, Scientific and Technical
Information Branch

1 Atch
AFDTC/PA Ltr, dtd 30 Jan 92

ERRATA

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE DEVELOPMENT TEST CENTER (AFSC)
EGLIN AIR FORCE BASE, FLORIDA 32542-5000

REPLY TO ATTN OF: PA (Jim Swinson, 882-3931) 30 January 1992

SUBJECT: Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

Virginia N. Pribyla

VIRGINIA N. PRIBYLA, Lt Col, USAF
Chief of Public Affairs

AFDTC/PA 92-039