<table>
<thead>
<tr>
<th>AD NUMBER</th>
<th>LIMITATION CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADB120256</td>
<td>TO: Approved for public release; distribution is unlimited.</td>
</tr>
</tbody>
</table>

**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
**Common Ada Missile Package (CAMP) Project: Missile Software Parts, Vol 9:**

**Detail Design Documents (Vol 7-12)**

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. Two volumes of this document are 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.
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)
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

Distribution authorized to U.S. Government agencies and their contractors only; this report documents test and evaluation; distribution limitation applied July 1987. Other requests for this document must be referred to the Air Force Armament Laboratory (FXG) Eglin Air Force Base, Florida 32542-5434.

DESTRUCTION NOTICE – For classified documents, follow the procedures in DoD 5220.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

WARNING: This document contains technical data whose export is restricted by the Arms Export Control Act (Title 22, U.S.C., Sec. 2751, et seq.) or the Export Administration Act of 1979, as amended (Title 50, U.S.C., App. 2401, et seq.). Violations of these export laws are subject to severe criminal penalties. Disseminate in accordance with the provisions of AFR 80-34.
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_MatrixTranspose_Multiply_Unrestricted is separate;

function Matrix_MatrixTranspose_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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dot Product</td>
<td>R063</td>
</tr>
<tr>
<td>Vector Length</td>
<td>R104</td>
</tr>
<tr>
<td>&quot;+&quot;</td>
<td>R061</td>
</tr>
<tr>
<td>&quot;-&quot;</td>
<td>R062</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Elements</td>
<td>floating point type</td>
<td>Type of elements to be contained in vector type defined by this package</td>
</tr>
<tr>
<td>Vector Elements_</td>
<td>floating point type</td>
<td>Resulting type from the operation</td>
</tr>
<tr>
<td>Squared</td>
<td></td>
<td>Vector Elements * Vector Elements; used for result of a dot product operation</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

Subprograms:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Used to define the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vector_Elements * Vector_Elements := Vector_Elements_Squared</td>
</tr>
<tr>
<td>SqRt</td>
<td>function</td>
<td>Square root function taking an object of type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vector_Elements_Squared and returning an object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of type Vector_Elements</td>
</tr>
</tbody>
</table>

3.3.6.2.9.1.4 LOCAL DATA

Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Unconstrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>

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:

```plaintext
dependent (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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Vectors</td>
<td>In</td>
<td>One of the vectors to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Vectors</td>
<td>In</td>
<td>Second vector to be added</td>
</tr>
</tbody>
</table>

3.3.6.2.9.1.10.1.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vectors</td>
<td>Vector being calculated and returned</td>
</tr>
<tr>
<td>L_Index</td>
<td>Indices</td>
<td>Index into Left vector</td>
</tr>
<tr>
<td>R_Index</td>
<td>Indices</td>
<td>Index into Right vector</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating</td>
<td>Type of elements to be contained in vector</td>
</tr>
<tr>
<td>Indices</td>
<td>point type</td>
<td>type defined by this package</td>
</tr>
<tr>
<td></td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Unconstrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>exception</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.1.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the lengths of the input vectors are not the same</td>
</tr>
</tbody>
</table>
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.

This part meets CAMP requirement R062.

None.

The following table describes this part's formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Vectors</td>
<td>In</td>
<td>Vector to act as the minuend</td>
</tr>
<tr>
<td>Right</td>
<td>Vectors</td>
<td>In</td>
<td>Vector to act as the subtrahend</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vectors</td>
<td>Vector being calculated and returned</td>
</tr>
<tr>
<td>L_Index</td>
<td>Indices</td>
<td>Index into Left vector</td>
</tr>
<tr>
<td>R_Index</td>
<td>Indices</td>
<td>Index into Right vector</td>
</tr>
</tbody>
</table>

Not applicable.

The following describes the processing performed by this part:

function "-" (Left : Vectors;
Right : Vectors) return Vectors is

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating</td>
<td>Type of elements to be contained in vector</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Unconstrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Vectors</td>
<td>In</td>
<td>Vector for which a length is desired</td>
</tr>
</tbody>
</table>

3.3.6.2.9.1.10.3.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temp</td>
<td>Vector_Elements_Squared</td>
<td>Used for intermediate calculations</td>
</tr>
</tbody>
</table>

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

    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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements to be contained in vector type defined by this package</td>
</tr>
<tr>
<td>Vector_Elements_Squared</td>
<td>floating point type</td>
<td>Resulting type from the operation</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Unconstrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>
3.3.6.2.9.1.10.4.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vector_Elements_Squared</td>
<td>Result of the dot product operation</td>
</tr>
<tr>
<td>L_Index</td>
<td>Indices</td>
<td>Index into Left vector</td>
</tr>
<tr>
<td>R_Index</td>
<td>Indices</td>
<td>Index into Right vector</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements to be contained in vector type defined by this package</td>
</tr>
<tr>
<td>Vector_Elements_Squared</td>
<td>floating point type</td>
<td>Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Unconstrained:
<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Unconstrained, one-dimensional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>array of elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>exception</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Used to define the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vector_Elements * Vector_Elements :=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vector_Elements_Squared</td>
</tr>
</tbody>
</table>

3.3.6.2.9.1.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the lengths of the two input vectors are not the same</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;+&quot; (matrices + matrices)</td>
<td>R079</td>
</tr>
<tr>
<td>&quot;-&quot; (matrices - matrices)</td>
<td>R080</td>
</tr>
<tr>
<td>&quot;+&quot; (matrices + elements)</td>
<td>R075</td>
</tr>
<tr>
<td>&quot;-&quot; (matrices - elements)</td>
<td>R076</td>
</tr>
<tr>
<td>Set_to_Identity_Matrix</td>
<td>R155</td>
</tr>
<tr>
<td>Set_to_Zero_Matrix</td>
<td>R156</td>
</tr>
<tr>
<td>&quot;**&quot;</td>
<td>R077</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.4 LOCAL DATA

Data types:

The following data type was previously defined at the package specification level:
<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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

dend 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>First matrix to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Second matrix to be added</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.1.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Result of adding the two input matrices</td>
</tr>
<tr>
<td>L_Col</td>
<td>Col_Indices</td>
<td>Left column index</td>
</tr>
<tr>
<td>L_Row</td>
<td>Row_Indices</td>
<td>Left row index</td>
</tr>
<tr>
<td>R_Col</td>
<td>Col_Indices</td>
<td>Right column index</td>
</tr>
<tr>
<td>R_Row</td>
<td>Row_Indices</td>
<td>Right row index</td>
</tr>
</tbody>
</table>

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:

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

   --declaration section--
   --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 L'LENGTH(l) = R'LENGTH(l) and
L'LENGTH(2) = R'LENGTH(2) then

L_Row := L'FIRST(l);
R_Row := R'FIRST(l);
Row_Loop:
  loop
    L_Col := L'FIRST(2);
    R_Col := R'FIRST(2);
    Col_Loop:
      loop
        Answer(L_Row, L_Col) := L(L_Row, L_Col) +
                                R(R_Row, R_Col);
        exit Col_Loop when L_Col = L'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 = L'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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the respective lengths of the first and second dimensions of the input matrices are not equal</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to act as the minuend</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be used as the subtrahend</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.2.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Result of adding the two input matrices</td>
</tr>
<tr>
<td>L_Col</td>
<td>Col_Indices</td>
<td>Left column index</td>
</tr>
<tr>
<td>L_Row</td>
<td>Row_Indices</td>
<td>Left row index</td>
</tr>
<tr>
<td>R_Col</td>
<td>Col_Indices</td>
<td>Right column index</td>
</tr>
<tr>
<td>R_Row</td>
<td>Row_Indices</td>
<td>Right row index</td>
</tr>
</tbody>
</table>

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
```
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 --begin function "-

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
ansver(L_Rov, L_Col) := Left(L_Row, L_Col) - 
Right(R_6ow, 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the respective lengths of the first and second dimensions of the input matrices are not the same</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Addend</td>
<td>Elements</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.3.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Scaled matrix</td>
</tr>
</tbody>
</table>

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:

```algebra
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Subtrahend</td>
<td>Elements</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.4.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Scaled matrix</td>
</tr>
</tbody>
</table>

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:

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

-- declaration section
--

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

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>
The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Unconstrained:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Matrix to be made into an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.5.4 LOCAL DATA

Data objects:

The following data objects are maintained local to this part.
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Matrix to be zeroed out</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matr.ces</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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 \( m \times n \) matrix by an \( n \times p \) matrix, returning an \( m \times p \) matrix. The type of elements in each of the three matrices is the same.

The values in the result matrix are defined as:

\[
 a(m,p) := b(m,n) \times c(n,p) 
\]

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>m x n matrix to act as multiplicand</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>n x p matrix to act as multiplier</td>
</tr>
</tbody>
</table>

3.3.6.2.9.2.10.7.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of multiplying two input matrices</td>
</tr>
<tr>
<td>M</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into rows of left and answer matrices</td>
</tr>
<tr>
<td>N_Left</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Index into columns of left matrix</td>
</tr>
<tr>
<td>N_Right</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into columns of right matrix</td>
</tr>
<tr>
<td>P</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Index into columns of left and answer matrices</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

Subprograms:

The following table summarizes the generic subroutines available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:
3.3.6.2.9.2.10.7.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the inner dimensions of the input matrices do not have the same length</td>
</tr>
</tbody>
</table>

3.3.6.2.9.3 DYNAMICALLY SPARSE_MATRIX_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN

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:
3.3.6.2.9.3.4 LOCAL DATA

Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix being made into an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.3.10.1.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Marker</td>
<td>Col_Indices</td>
<td>Index into second dimension of input matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>Index into first dimension of input matrix</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.3.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be zeroed out</td>
</tr>
</tbody>
</table>

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:

```vhdl
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:
The following types are defined in the package specification for DynamicallySparse_Matrix_Operations_Unconstrained:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to which is added an identity matrix</td>
</tr>
</tbody>
</table>
3.3.6.2.9.3.10.3.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding an identity matrix</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>to the input matrix</td>
</tr>
<tr>
<td>Col_Marker</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index</td>
</tr>
</tbody>
</table>

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:

```plaintext
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;
    end loop;
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Square matrix to be subtracted from an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.3.10.4.4 LOCAL DATA

Data objects:
### 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:

```plaintext
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:

<table>
<thead>
<tr>
<th></th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.3.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>Sparse matrix to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Sparse matrix to be added</td>
</tr>
</tbody>
</table>
3.3.6.2.9.3.10.5.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding two input matrices</td>
</tr>
<tr>
<td>L_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into left matrix</td>
</tr>
<tr>
<td>L_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into left matrix</td>
</tr>
<tr>
<td>R_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into right matrix</td>
</tr>
<tr>
<td>R_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into right matrix</td>
</tr>
</tbody>
</table>

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:

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

3.3.6.2.9.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OFANCESTRAL 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:
### Elements

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

#### Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

### LIMITATIONS

3.3.6.2.9.3.10.5.8

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if both matrices are not m x n matrices</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>Sparse matrix to be treated as the minuend</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Sparse matrix to be treated as the subtrahend</td>
</tr>
</tbody>
</table>

3.3.6.2.9.3.10.6.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of subtracting two input matrices</td>
</tr>
<tr>
<td>L_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into left matrix</td>
</tr>
<tr>
<td>L_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into left matrix</td>
</tr>
<tr>
<td>R_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into right matrix</td>
</tr>
<tr>
<td>R_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into right matrix</td>
</tr>
</tbody>
</table>

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:

```haskell
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;

   end loop Row_Loop;

else

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>
3.3.6.2.9.3.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if both matrices are not m x n matrices</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap_Row</td>
<td>function</td>
<td>Takes a column as input, and returns the corresponding row entry</td>
</tr>
<tr>
<td>Swap_Col</td>
<td>function</td>
<td>Takes a row as input, and returns the corresponding column entry</td>
</tr>
</tbody>
</table>

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, 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in the half storage matrix and in the Slices array</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>Rov_Indices</td>
<td>discrete</td>
<td>Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored</td>
</tr>
<tr>
<td>Col_Slices</td>
<td>array</td>
<td>Data type defining a column slice of a matrix</td>
</tr>
<tr>
<td>Rov_Slices</td>
<td>array</td>
<td>Data type defining a row slice of a matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Index_Arrays</td>
<td>array</td>
<td>Array of integers indexed by Indices used to set up column markers into Matrices array</td>
</tr>
<tr>
<td>Rov_Index_Arrays</td>
<td>array</td>
<td>Array of integers indexed by Indices used to set up row markers into Matrices array</td>
</tr>
</tbody>
</table>

Data objects:

The following table describes the data objects defined in this part's package specification:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>---</td>
<td>Number of stored values from the half-storage matrix; the number of elements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stored in a half-storage matrix with n rows elements is n(n+1)/2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index_</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td></td>
<td>Arrays</td>
<td></td>
</tr>
<tr>
<td>Row_Marker</td>
<td>Row_Index_</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
<tr>
<td></td>
<td>Arrays</td>
<td></td>
</tr>
<tr>
<td>Local_Matrices</td>
<td>Matrices</td>
<td>Pre-initialized identity matrix</td>
</tr>
<tr>
<td>Identity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local_Matrices</td>
<td>Matrices</td>
<td>Pre-initialized zero matrix</td>
</tr>
<tr>
<td>Zero_Matrics</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\text{Full Storage}(i,j) \leftrightarrow \text{Half Storage} (\text{Row Marker}(i) + \text{Col Offset}(j));
\]

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

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

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

  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'SUCCE(Row_Indices'FIRST) ..
            Row_Indices'LAST loop

          Row_Marker(Index) := 1;

          if Index /= Row_Indices'LAST then
            Row_Marker(Index + 1) := 1;
          end if;

        while Count <= Row_Slices'LENGTH loop
          Row_Marker(Index) := 1;

          if Index /= Row_Indices'LAST then
            Row_Marker(Index + 1) := 1;
          end if;

          Count := Count + 1;
        end loop;

      end if;
    end begin;
end;
Row_Start = (Count \times (Count + 1) / 2) + 1;

Row Marker(Index) := Row_Start;
Local_Identity_Matrix(Row_Start-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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:
### Dimension_Error

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if an attempt is made to instantiated other than</td>
</tr>
<tr>
<td></td>
<td>a square matrix</td>
</tr>
</tbody>
</table>

#### 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 as 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>In</td>
<td>Row to be converted to a column entry</td>
</tr>
</tbody>
</table>

##### 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
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored</td>
</tr>
</tbody>
</table>

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 as 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>In</td>
<td>Column to be converted to a row entry</td>
</tr>
</tbody>
</table>

3.3.6.2.9.4.10  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:

```haskell
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_Slice</td>
<td>Slices</td>
<td>In</td>
<td>Row of values to be placed in the half-storage matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row</td>
<td>In</td>
<td>Indicates which row of values this is</td>
</tr>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Half-storage matrix to be initialized</td>
</tr>
</tbody>
</table>

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:

```plaintext
procedure Initialize (Row_Slice : in Row_Slices;
                      Row       : in Row_Indices;
                      Matrix    : out Matrices) is
```
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:
The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

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

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap_Row</td>
<td>function</td>
<td>Takes a column as input, and returns the corresponding row entry</td>
</tr>
<tr>
<td>Swap_Col</td>
<td>function</td>
<td>Takes a row as input, and returns the corresponding column entry</td>
</tr>
</tbody>
</table>
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:

```pascal
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Identity Matrix</td>
<td>Matrices</td>
<td>Pre-initialized identity matrix</td>
</tr>
</tbody>
</table>

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:

```pascal
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local_Zero</td>
<td>Matrices</td>
<td>Pre-initialized zero matrix</td>
</tr>
<tr>
<td>Matrix</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New_Value</td>
<td>Elements</td>
<td>In</td>
<td>New value to be placed in Matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>In</td>
<td>Row where New Value is to be placed</td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>In</td>
<td>Column where New Value is to be placed</td>
</tr>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Half-storage matrix to be updated</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the half storage matrix and in the Slices array</td>
</tr>
<tr>
<td>Col</td>
<td>discrete type</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>Indices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rov</td>
<td>discrete type</td>
<td>Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored</td>
</tr>
<tr>
<td>Indices</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>
Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index_Arrays</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td>Row_Marker</td>
<td>Row_Index_Arrays</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap_Row</td>
<td>function</td>
<td>Takes a column as input, and returns the corresponding row entry</td>
</tr>
<tr>
<td>Swap_Col</td>
<td>function</td>
<td>Takes a row as input, and returns the corresponding column entry</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Half-storage matrix containing desired element</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>In</td>
<td>Row in which element is contained</td>
</tr>
</tbody>
</table>

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:

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

    Answer : Elements;

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the half storage matrix and in the Slices array</td>
</tr>
<tr>
<td>Col</td>
<td>discrete</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>Indices</td>
<td>type</td>
<td>Used to dimension row slices of the half storage matrix and to determine the</td>
</tr>
<tr>
<td>Row</td>
<td>discrete</td>
<td>number of elements which need to be stored</td>
</tr>
<tr>
<td>Indices</td>
<td>type</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index_Arrays</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td>Row_Marker</td>
<td>Row_Index_Arrays</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap_Row</td>
<td>function</td>
<td>Takes a column as input, and returns the corresponding row entry</td>
</tr>
<tr>
<td>Swap_Col</td>
<td>function</td>
<td>Takes a row as input, and returns the corresponding column entry</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Half-storage matrix containing values to be retrieved</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>In</td>
<td>Indicates which row of values is desired</td>
</tr>
</tbody>
</table>

3.3.6.2.9.4.10.8.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Row_Slices</td>
<td>N/A</td>
<td>Requested row of values</td>
</tr>
</tbody>
</table>

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:

```plaintext
function Row_Slice (Matrix : Matrices;
                    Row   : Row Indices) return Row_Slices is
  begin
    --          --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 package 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to dimension row slices of the half storage matrix and to determine</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>the number of elements which need to be stored</td>
</tr>
<tr>
<td>Row_Slices</td>
<td>array</td>
<td>Data type defining a row slice of a matrix</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index_</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td></td>
<td>Arrays</td>
<td></td>
</tr>
<tr>
<td>Row_Marker</td>
<td>Row_Index_</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
<tr>
<td></td>
<td>Arrays</td>
<td></td>
</tr>
</tbody>
</table>

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:
### 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Half-storage matrix from which a column of values is to be retrieved</td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>In</td>
<td>Indicates which column of values is desired</td>
</tr>
</tbody>
</table>

#### 3.3.6.2.9.4.10.9.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Col_Slices</td>
<td>N/A</td>
<td>Column's worth of values</td>
</tr>
</tbody>
</table>
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:
### Elements
- **Type**: floating point type
- **Description**: Data type of elements in the half storage matrix and in the Slices array

### Col Indices
- **Type**: discrete type
- **Description**: Used to dimension column slices

### Row Indices
- **Type**: discrete type
- **Description**: Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

### Col Slices
- **Type**: array
- **Description**: 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**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

---

**Data objects:**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index_ ...</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td>Row_Marker</td>
<td>Row_Index_ ...</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
</tbody>
</table>

---

**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**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap_Row</td>
<td>function</td>
<td>Takes a column as input, and returns the corresponding row entry</td>
</tr>
<tr>
<td>Swap_Col</td>
<td>function</td>
<td>Takes a row as input, and returns the corresponding column entry</td>
</tr>
</tbody>
</table>
3.3.6.2.9.4.10.8 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be added to an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.4.10.10.4 LOCAL DATA
Data objects:
The following table summarizes the data objects maintained by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Result of adding input matrix to identity matrix</td>
</tr>
</tbody>
</table>

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'SUC(Row_Indices'FIRST) ..
         Row_Indices'LAST loop
      begin
        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:
The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the half storage matrix and in the Slices array</td>
</tr>
<tr>
<td>Rov_Indices</td>
<td>discrete type</td>
<td>Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>---</td>
<td>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</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index_Arrays</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td>Rov_Marker</td>
<td>Rov_Index_Arrays</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
</tbody>
</table>

3.3.6.2.9.4.10.10.8 LIMITATIONS

None.
3.3.6.2.9.4.10.11 SUBTRACT_FROM_ITY 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be subtracted from an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.4.10.11.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Result of subtracting input matrix from an identity matrix</td>
</tr>
</tbody>
</table>

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
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:
lor 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:

---

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point</td>
<td>Data type of elements in the half storage matrix and in the Slices array</td>
</tr>
<tr>
<td></td>
<td>discrete type</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>Row Indices</td>
<td>discrete type</td>
<td>Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>---</td>
<td>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</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index_ Arrays</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td>Row_Marker</td>
<td>Row_Index_ Arrays</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>First matrix to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Second matrix to be added</td>
</tr>
</tbody>
</table>

3.3.6.2.9.4.10.12.4 LOCAL DATA

Data objects:

The following data objects are maintained by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Result of adding the two input matrices</td>
</tr>
</tbody>
</table>

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:

```plaintext
function "+" (Left : Matrices;
             Right : Matrices) return Matrices is
```
```plaintext
-- -----------------------------
-- --declaration section
```
Answer : Matrices;

begin function "+

begin

Process:
  for Index in Entry_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 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the half storage matrix and in the Slices array</td>
</tr>
<tr>
<td>ColIndices</td>
<td>discrete type</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>RowIndices</td>
<td>discrete type</td>
<td>Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>---</td>
<td>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</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index_</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td>Row_Marker</td>
<td>Row_Index_</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
<tr>
<td></td>
<td>Arrays</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap_Row</td>
<td>function</td>
<td>Takes a column as input, and returns the corresponding row entry</td>
</tr>
<tr>
<td>Swap_Col</td>
<td>function</td>
<td>Takes a row as input, and returns the corresponding column entry</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be subtracted from</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be subtracted from Left</td>
</tr>
</tbody>
</table>

3.3.6.2.9.4.10.13.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of subtracting Right input matrix from Left input matrix</td>
</tr>
</tbody>
</table>

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:

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

-- declaration section

Answer : Matrices;
```

-- ----------------------
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 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the half storage matrix and in the Slices array</td>
</tr>
<tr>
<td>ColIndices</td>
<td>discrete type</td>
<td>Used to dimension column slices</td>
</tr>
<tr>
<td>RovIndices</td>
<td>discrete type</td>
<td>Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>---</td>
<td>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</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Offset</td>
<td>Col_Index</td>
<td>Used to determine the offset of a column index from the first column index</td>
</tr>
<tr>
<td>Row_Marker</td>
<td>Row_Index</td>
<td>Used to marker where in Matrices a particular row begins</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap_Row</td>
<td>function</td>
<td>Takes a column as input, and returns the corresponding row entry</td>
</tr>
<tr>
<td>Swap_Col</td>
<td>function</td>
<td>Takes a row as input, and returns the corresponding column entry</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td></td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td></td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.4 LOCAL DATA

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid_Index</td>
<td>Indicates an attempt was made to access an element beyond the dimensions of the array</td>
</tr>
</tbody>
</table>

Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
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:

```plaintext
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:
### LOCAL DATA

None.

### PROCESS CONTROL

Not applicable.

### PROCESSING

The following describes the processing performed by this part:

```plaintext
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; 

eンド if; 

eンド 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 
or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid_Index</td>
<td>Indicates an attempt was made to access an element beyond the dimensions of the array</td>
</tr>
</tbody>
</table>

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid_Index</td>
<td>Raised if an attempt is made to place an element outside the bounds of the input array</td>
</tr>
<tr>
<td>Dimension_Error</td>
<td>Raised if the input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Matrix to be made into an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.10.2.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of input matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of input matrix</td>
</tr>
</tbody>
</table>

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:

```plaintext
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 or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension Error</td>
<td>Raised if input matrix is a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Matrix to be zeroed out</td>
</tr>
</tbody>
</table>
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:

```plaintext
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 or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>point type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be added to an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.10.4.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding identity matrix to input</td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of matrices</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of matrices</td>
</tr>
</tbody>
</table>

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

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 or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be subtracted from an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.10.5.4 LOCAL DATA

Data objects:

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

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

```

Subtract_Elements_From_Zero:

---

-- 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;
S_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
or 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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>First matrix to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Second matrix to be added</td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.10.6.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding input matrices</td>
</tr>
<tr>
<td>Row Count, Col Count</td>
<td>Positive</td>
<td>N/A</td>
<td>Used to keep track of the number of rows and columns which have been handled</td>
</tr>
<tr>
<td>L Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into left matrix</td>
</tr>
<tr>
<td>L Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into left matrix</td>
</tr>
<tr>
<td>R Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into right matrix</td>
</tr>
<tr>
<td>R Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into right matrix</td>
</tr>
<tr>
<td>S_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>New column index after row and column have been swapped, i.e. (i,j) -&gt; (j,i)</td>
</tr>
<tr>
<td>S_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>New row index after row and column have been swapped</td>
</tr>
</tbody>
</table>

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:

```plaintext
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 or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>point type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_Unconstrained:
<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.5.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if input matrices are not both m x m matrices</td>
</tr>
</tbody>
</table>

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:
### 3.3.6.2.9.5.10.7.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding input matrices</td>
</tr>
<tr>
<td>Row_Cnt,</td>
<td>Matrices</td>
<td>N/A</td>
<td>Used to keep track of the number of rows and columns which have been handled</td>
</tr>
<tr>
<td>COL_Cnt</td>
<td>Positive</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>L_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into left matrix</td>
</tr>
<tr>
<td>L_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into left matrix</td>
</tr>
<tr>
<td>R_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into right matrix</td>
</tr>
<tr>
<td>R_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into right matrix</td>
</tr>
<tr>
<td>S_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>New column index after row and column have been swapped, i.e. (i,j) -&gt; (j,i)</td>
</tr>
<tr>
<td>S_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>New row index after row and column have been swapped</td>
</tr>
</tbody>
</table>

### 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:

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

-- declaration section

Answer : Matrices(Left'RANGE(1), Left'RANGE(2));
Col_Cnt : POSITIVE;
Row_Cnt : 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 or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>point type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Unconstrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

### 3.3.6.2.9.5.10.7.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if input matrices are not both m x m matrices</td>
</tr>
</tbody>
</table>

### 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:
### 3.3.6.2.9.6.4 LOCAL DATA

#### Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid_Index</td>
<td>Indicates an attempt was made to access an element not on the diagonal</td>
</tr>
</tbody>
</table>

#### Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal_</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal_matrices</td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diagonal_</td>
<td>N/A</td>
<td>Vector representation of a matrix where all but the diagonal elements equal zero</td>
</tr>
<tr>
<td>Matrices</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_Markers</td>
<td>N/A</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
<tr>
<td>Col_Markers</td>
<td>N/A</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>Number of diagonal elements in the array</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_Marker</td>
<td>Row_Markers</td>
<td>Array of pointers into Diagonal Matrices</td>
</tr>
<tr>
<td>Col_Marker</td>
<td>Col_Markers</td>
<td>Array of pointers into Diagonal Matrices</td>
</tr>
<tr>
<td>Row_Minus_Col_Indices_Pos_First</td>
<td>INTEGER</td>
<td>Preinitialized value of: Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)</td>
</tr>
<tr>
<td>Local_Identity_Matrix</td>
<td>Diagonal Matrices</td>
<td>Pre-Initialied identity matrix</td>
</tr>
<tr>
<td>Local_Zero_Matrix</td>
<td>Diagonal Matrices</td>
<td>Pre-initialized zero matrix</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Count</td>
<td>Positive</td>
<td>N/A</td>
<td>Counts the number of columns</td>
</tr>
<tr>
<td>Row_Count</td>
<td>Positive</td>
<td>N/A</td>
<td>Counts the number of rows</td>
</tr>
</tbody>
</table>

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 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the lengths of Row_Indices and Col_Indices are not the same</td>
</tr>
</tbody>
</table>

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:

```plaintext
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 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
</tbody>
</table>

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:
<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal matrices</td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diagonal</td>
<td>N/A</td>
<td>Vector representation of a matrix where all</td>
</tr>
<tr>
<td>Matrices</td>
<td></td>
<td>but the diagonal elements equal zero</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local_Identity_Matrix</td>
<td>Diagonal_ Matrices</td>
<td>Pre-initialized identity matrix</td>
</tr>
</tbody>
</table>

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
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal Range</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal_matrices</td>
</tr>
<tr>
<td>Diagonal Matrices</td>
<td>N/A</td>
<td>Vector representation of a matrix where all but the diagonal elements equal zero</td>
</tr>
</tbody>
</table>

Data objects:

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

None.

### CHANGE_ELEMENT UNIT DESIGN (CATALOG #P411-0)

This procedure changes a single element of a diagonal matrix.

### REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

### LOCAL ENTITIES DESIGN

None.

### INPUT/OUTPUT

**FORMAL PARAMETERS:**

The following table describes this part's formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New_Value</td>
<td>Elements</td>
<td>In</td>
<td>New value to be placed in matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row Indices</td>
<td>In</td>
<td>Row where value is to be placed</td>
</tr>
<tr>
<td>Col</td>
<td>Col Indices</td>
<td>In</td>
<td>Column where value is to be placed</td>
</tr>
<tr>
<td>Matrix</td>
<td>Diagonal Matrices</td>
<td>Out</td>
<td>Diagonal matrix to be updated</td>
</tr>
</tbody>
</table>

### LOCAL DATA

None.

### 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal_</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal_matrices</td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diagonal_</td>
<td>N/A</td>
<td>Vector representation of a matrix where all</td>
</tr>
<tr>
<td>Matrices</td>
<td></td>
<td>but the diagonal elements equal zero</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_Markers</td>
<td>N/A</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
<tr>
<td>Col_Markers</td>
<td>N/A</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid_Index</td>
<td>Indicates an attempt was made to access an element not on the diagonal</td>
</tr>
</tbody>
</table>

3.3.6.2.9.6.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid_Index</td>
<td>Raised if in Row any Col indices do not fall on the diagonal</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Diagonal Matrices</td>
<td>In</td>
<td>Diagonal matrix containing element desired</td>
</tr>
<tr>
<td>Row</td>
<td>Row Indices</td>
<td>In</td>
<td>Row containing element desired</td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>In</td>
<td>Column containing element desired</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal_</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal_matrices</td>
</tr>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Vector representation of a matrix where all but the diagonal elements equal zero</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_Markers</td>
<td>N/A</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
<tr>
<td>Col_Markers</td>
<td>N/A</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Row_Marker</th>
<th>Row_Markers</th>
<th>Array of pointers into Diagonal Matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Marker</td>
<td>Col_Markers</td>
<td>Array of pointers into Diagonal Matrices</td>
</tr>
</tbody>
</table>

#### 3.3.6.2.9.6.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid_Index</td>
<td>Raised if Row and Col indices do not fall on the diagonal!</td>
</tr>
</tbody>
</table>

#### 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Diagonal</td>
<td>In</td>
<td>Diagonal matrix containing row desired</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>In</td>
<td>Row desired</td>
</tr>
</tbody>
</table>

#### 3.3.6.2.9.6.10.5.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Spot</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Index into Answer</td>
</tr>
<tr>
<td>Answer</td>
<td>Row_Slices</td>
<td>N/A</td>
<td>Row of values</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Slices</td>
<td>array</td>
<td>One-dimensional array of row Elements</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal_</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal matrices</td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diagonal_Matrices</td>
<td>N/A</td>
<td>Vector representation of a matrix where all but the diagonal elements equal zero</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_Markers</td>
<td>N/A</td>
<td>N/A</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Marker</td>
<td>Col_Markers</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
<tr>
<td>Row_Minus_Col_Indices_Pos_First</td>
<td>INTEGER</td>
<td>Preinitialized value of: Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Diagonal_</td>
<td>In</td>
<td>Diagonal matrix containing desired column of values</td>
</tr>
<tr>
<td></td>
<td>Matrices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>In</td>
<td>Indicates which column is desired</td>
</tr>
</tbody>
</table>

3.3.6.2.9.6.10.6.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Col_Slices</td>
<td>N/A</td>
<td>Column of values from input matrix</td>
</tr>
<tr>
<td>Row_Spot</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into Answer</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements'</td>
<td>floating point type</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Col_Slices</td>
<td>array</td>
<td>One-dimensional array of column Elements</td>
</tr>
</tbody>
</table>
The following table summarizes the types required by this part and defined in
the package specification of Diagonal_Matrix_Operations:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal matrices</td>
</tr>
<tr>
<td>Range</td>
<td>N/A</td>
<td>Vector representation of a matrix where all but the diagonal elements equal zero</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Markers</td>
<td>N/A</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col_Marker</td>
<td>Col_Markers</td>
<td>Array of pointers into Diagonal_Matrices</td>
</tr>
<tr>
<td>Row_Minus_Col_Indices_Pos_First</td>
<td>INTEGER</td>
<td>Preinitialized value of: Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Diagonal_Matrices</td>
<td>In</td>
<td>Diagonal matrix to be added to an identity matrix.</td>
</tr>
</tbody>
</table>

3.3.6.2.9.6.10.7.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Diagonal_Matrices</td>
<td>N/A</td>
<td>Result of performing addition</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal_Range</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal_matrices</td>
</tr>
<tr>
<td>Diagonal_Matrices</td>
<td>N/A</td>
<td>Vector representation of a matrix where all but the diagonal elements equal zero</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>Number of diagonal elements in the array</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Diagonal_Matrices</td>
<td>In</td>
<td>Diagonal matrix to be subtracted from an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.6.10.8.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Diagonal_Matrices</td>
<td>N/A</td>
<td>Result of performing the subtraction</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
</tbody>
</table>

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:
### Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>Number of diagonal elements in the array</td>
</tr>
</tbody>
</table>

#### 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Diagonal_Matrices</td>
<td>In</td>
<td>First diagonal matrix to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Diagonal_Matrices</td>
<td>In</td>
<td>Second diagonal matrix to be added</td>
</tr>
</tbody>
</table>
3.3.6.2.9.6.10.9.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Diagonal_Matrices</td>
<td>N/A</td>
<td>Result of performing the addition</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal_(j) Range</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal_matrices</td>
</tr>
<tr>
<td>Diagonal_(j) Matrices</td>
<td>N/A</td>
<td>Vector representation of a matrix where all but the diagonal elements equal zero</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>Number of diagonal elements in the array</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Diagonal_Matrices</td>
<td>In</td>
<td>Diagonal matrix to be subtracted from</td>
</tr>
<tr>
<td>Right</td>
<td>Diagonal_Matrices</td>
<td>In</td>
<td>Diagonal matrix to be treated as the subtrahend</td>
</tr>
</tbody>
</table>

3.3.6.2.9.6.10.10.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Diagonal_Matrices</td>
<td>N/A</td>
<td>Result of performing the subtraction</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in the exported matrix type, as well as the imported array types</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to dimension imported and exported arrays</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal_Matrices</td>
<td>N/A</td>
<td>Vector representation of a matrix where all but the diagonal elements equal zero</td>
</tr>
<tr>
<td>Diagonal_Range</td>
<td>1..Entry_Count</td>
<td>Used to dimension diagonal_matrices</td>
</tr>
</tbody>
</table>

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry_Count</td>
<td>Positive</td>
<td>Number of diagonal elements in the array</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>R065</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>R066</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating</td>
<td>Type of elements in a vector;</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td>Elements1 := Elements2 * Scalars</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating</td>
<td>Type of elements in a vector;</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td>Elements2 := Elements1 / Scalars</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating</td>
<td>Type of value to be used for multiplying and dividing</td>
</tr>
<tr>
<td>Indices1</td>
<td>discrete</td>
<td>Used to dimension Vectors1</td>
</tr>
<tr>
<td>Indices2</td>
<td>discrete</td>
<td>Used to dimension Vectors2</td>
</tr>
<tr>
<td>Vectors1</td>
<td>array</td>
<td>An array of Elements1</td>
</tr>
<tr>
<td>Vectors2</td>
<td>array</td>
<td>An array of Elements2</td>
</tr>
</tbody>
</table>

Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Used to define the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elements1 := Elements2 * Scalars</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>function</td>
<td>Used to define the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elements2 := Elements1 / Scalars</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Vectors2</td>
<td>In</td>
<td>Vector to be scaled</td>
</tr>
<tr>
<td>Multiplier</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.7.10.1.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vectors1</td>
<td>Scaled vector</td>
</tr>
<tr>
<td>A_Index</td>
<td>Indices1</td>
<td>Index into answer array</td>
</tr>
<tr>
<td>V_Index</td>
<td>Indices2</td>
<td>Index into input vector</td>
</tr>
</tbody>
</table>
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(Vectors1'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 = Vectors1'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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in a vector; Elements1 := Elements2 * Scalars</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in a vector; Elements2 := Elements1 / Scalars</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Type of value to be used for multiplying and dividing</td>
</tr>
<tr>
<td>Indices1</td>
<td>discrete type</td>
<td>Used to dimension Vectors1</td>
</tr>
<tr>
<td>Indices2</td>
<td>discrete type</td>
<td>Used to dimension Vectors2</td>
</tr>
<tr>
<td>Vectors1</td>
<td>array</td>
<td>An array of Elements1</td>
</tr>
<tr>
<td>Vectors2</td>
<td>array</td>
<td>An array of Elements2</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Used to define the operation Elements1 := Elements2 * Scalars</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Vectors1</td>
<td>In</td>
<td>Vector to be scaled</td>
</tr>
<tr>
<td>Divisor</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.7.10.2.4 LOCAL DATA

Data objects:

The following describes the local data maintained by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vectors2</td>
<td>Scaled vector</td>
</tr>
<tr>
<td>A_Index</td>
<td>Indices2</td>
<td>Index into answer vector</td>
</tr>
<tr>
<td>V_Index</td>
<td>Indices1</td>
<td>Index into input vector</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating</td>
<td>Type of elements in a vector;</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td>Elements1 := Elements2 * Scalars</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating</td>
<td>Type of elements in a vector;</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td>Elements2 := Elements1 / Scalars</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating</td>
<td>Type of value to be used for multiplying and dividing</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td></td>
</tr>
<tr>
<td>Indices1</td>
<td>discrete</td>
<td>Used to dimension Vectors1</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>Indices2</td>
<td>discrete</td>
<td>Used to dimension Vectors2</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>Vectors1</td>
<td>array</td>
<td>An array of Elements1</td>
</tr>
<tr>
<td>Vectors2</td>
<td>array</td>
<td>An array of Elements2</td>
</tr>
</tbody>
</table>

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:
### LIMITATIONS

None.

### 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.

### REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the parts in this LLCSC:

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/*&quot;</td>
<td>R073</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### LOCAL ENTITIES DESIGN

None.

### 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 | discrete | Used to dimension second dimension of Matrices1
| Indices1 | type |
| Row | discrete | Used to dimension first dimension of Matrices1
| Indices1 | type |
| Col | discrete | Used to dimension second dimension of Matrices2
| Indices2 | type |
| Row | discrete | Used to dimension first dimension of Matrices2
| Indices2 | type |
| 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

### LOCAL DATA

None.

### PROCESS CONTROL

Not applicable.

#### PROCESSING

The following describes the processing performed by this part:

```plaintext
separate (General Vector Matrix Algebra)
package body Matrix_Scalar_Operations_Unconstrained is
end Matrix_Scalar_Operations_Unconstrained;
```
### UTILIZATION OF OTHER ELEMENTS

None.

### LIMITATIONS

None.

### LLCSC DESIGN

None.

### UNIT DESIGN

#### "*" UNIT DESIGN (CATALOG #P426-0)

This function calculates a scaled matrix by multiplying each element of an input matrix by a scalar.

#### REQUIREMENTS ALLOCATION

This part meets CAMP requirement R073.

#### LOCAL ENTITIES DESIGN

None.

#### INPUT/OUTPUT

**FORMAL PARAMETERS:**

The following table describes this part’s formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matricesl</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Multiplier</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

#### LOCAL DATA

Data objects:

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

```pl
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;
    end loop
  end Loop
end begin;
```

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices2</td>
<td>Scaled matrix</td>
</tr>
<tr>
<td>A_Col</td>
<td>Col_Indices2</td>
<td>Index into second dimension of answer matrix</td>
</tr>
<tr>
<td>A_Row</td>
<td>Row_Indices2</td>
<td>Index into first dimension of answer matrix</td>
</tr>
<tr>
<td>M_Col</td>
<td>Col_Indices1</td>
<td>Index into second dimension of input matrix</td>
</tr>
<tr>
<td>M_Row</td>
<td>Row_Indices1</td>
<td>Index into first dimension of input matrix</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Data type of objects to be used as multipliers and divisors</td>
</tr>
<tr>
<td>Col_Indices1</td>
<td>discrete type</td>
<td>Used to dimension second dimension of Matrices1</td>
</tr>
<tr>
<td>Row_Indices1</td>
<td>discrete type</td>
<td>Used to dimension first dimension of Matrices1</td>
</tr>
<tr>
<td>Col_Indices2</td>
<td>discrete type</td>
<td>Used to dimension second dimension of Matrices2</td>
</tr>
<tr>
<td>Row_Indices2</td>
<td>discrete type</td>
<td>Used to dimension first dimension of Matrices2</td>
</tr>
<tr>
<td>Matrices1</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements1</td>
</tr>
<tr>
<td>Matrices2</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements2</td>
</tr>
</tbody>
</table>

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.
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices2</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Divisor</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.8.10.2.4 LOCAL DATA

Data objects:
The following table describes the data objects maintained by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices1</td>
<td>Scaled matrix</td>
</tr>
<tr>
<td>A_Col</td>
<td>Col_Indices1</td>
<td>Index into second dimension of answer matrix</td>
</tr>
<tr>
<td>A_Row</td>
<td>Row_Indices1</td>
<td>Index into first dimension of answer matrix</td>
</tr>
<tr>
<td>M_Col</td>
<td>Col_Indices2</td>
<td>Index into second dimension of input matrix</td>
</tr>
<tr>
<td>M_Row</td>
<td>Row_Indices2</td>
<td>Index into first dimension of input matrix</td>
</tr>
</tbody>
</table>

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;
      end loop
  end loop
end
```
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Data type of objects to be used as multipliers and divisors</td>
</tr>
<tr>
<td>Col</td>
<td>discrete type</td>
<td>Used to dimension second dimension of Matrices1</td>
</tr>
<tr>
<td>Indices1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row</td>
<td>discrete type</td>
<td>Used to dimension first dimension of Matrices1</td>
</tr>
<tr>
<td>Indices1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Col</td>
<td>discrete type</td>
<td>Used to dimension second dimension of Matrices2</td>
</tr>
<tr>
<td>Indices2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row</td>
<td>discrete type</td>
<td>Used to dimension first dimension of Matrices2</td>
</tr>
<tr>
<td>Indices2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrices1</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements1</td>
</tr>
<tr>
<td>Matrices2</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements2</td>
</tr>
</tbody>
</table>

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.
### LIMITATIONS

None.

### REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

### 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.

### 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function to define the operation [ \text{Elements}_1 \ast \text{Scalars} := \text{Elements}_2 ]</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>function</td>
<td>Function to define the operation [ \text{Elements}_2 \div \text{Scalars} := \text{Elements}_1 ]</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Elements1</td>
<td>floating point</td>
<td>Type of elements in Diagonal_Matrices1</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point</td>
<td>Type of elements in Diagonal_Matrices2</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point</td>
<td>Data type of scale factor</td>
</tr>
<tr>
<td>Diagonal_Rangels1</td>
<td>integer type</td>
<td>Used to dimension Diagonal_Matrices1</td>
</tr>
<tr>
<td>Diagonal_Ranges2</td>
<td>integer type</td>
<td>Used to dimension Diagonal_Matrices2</td>
</tr>
<tr>
<td>Diagonal_Matrices1</td>
<td>array</td>
<td>An array of Elements1</td>
</tr>
<tr>
<td>Diagonal_Matrices2</td>
<td>array</td>
<td>An array of Elements2</td>
</tr>
</tbody>
</table>

Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Multiplication operator defining the operation: Elements1 * Scalars = Elements2</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>function</td>
<td>Division operator defining the operation: Elements2 / Scalars = Elements1</td>
</tr>
</tbody>
</table>

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_Matrices_2Scalar_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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.9.8 LIMITATIONS

The following exceptions are raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the lengths of the two imported vector types are not of the same length</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Diagonal_Matrices1</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Multiplier</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.9.10.1.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Diagonal_Matrices2</td>
<td>N/A</td>
<td>Scaled diagonal matrix</td>
</tr>
<tr>
<td>Index1</td>
<td>Diagonal_Rangel</td>
<td>N/A</td>
<td>Index into input matrix</td>
</tr>
<tr>
<td>Index2</td>
<td>Diagonal_Range2</td>
<td>N/A</td>
<td>Index into output matrix</td>
</tr>
</tbody>
</table>

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:

```pl
function "*" (Matrix : Diagonal_Matrices1;
Multiplier : Scalars) return Diagonal_Matrices2 is

-- declaration section

Answer : Diagonal_Matrices2;
Index1 : Diagonal_Rangel;
Index2 : Diagonal_Range2;

-- begin function "*"
```

begin

Index1 := Diagonal_Rangel'FIRST;
Index2 := Diagonal_Range2'FIRST;
Process:
loop

Answer(Index2) := Matrix(Index1) * Multiplier;

exit Process when Index1 = Diagonal_Rangel'LAST;
Index1 := Diagonal_Rangel'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_Matrices

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in Diagonal_Matrices1</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in Diagonal_Matrices2</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Data type of scale factor</td>
</tr>
<tr>
<td>Diagonal_Rangel</td>
<td>integer type</td>
<td>Used to dimension Diagonal_Matrices1</td>
</tr>
<tr>
<td>Diagonal_Range2</td>
<td>integer type</td>
<td>Used to dimension Diagonal_Matrices2</td>
</tr>
<tr>
<td>Diagonal_Matrices1</td>
<td>array</td>
<td>An array of Elements1</td>
</tr>
<tr>
<td>Diagonal_Matrices2</td>
<td>array</td>
<td>An array of Elements2</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Diagonal_Matrices2</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Divisor</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.9.10.2.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Diagonal_Matrices1</td>
<td>N/A</td>
<td>Scaled diagonal matrix</td>
</tr>
<tr>
<td>Index1</td>
<td>Diagonal_Range1</td>
<td>N/A</td>
<td>Index into input matrix</td>
</tr>
<tr>
<td>Index2</td>
<td>Diagonal_Range2</td>
<td>N/A</td>
<td>Index into output matrix</td>
</tr>
</tbody>
</table>

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:

```plaintext
function "/" (Matrix : Diagonal_Matrices2;
   Divisor : Scalars) return Diagonal_Matrices1 is
```
Answer : Diagonal_Matrices1;
Index1 : Diagonal_Range1;
Index2 : Diagonal_Range2;

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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in Diagonal_Matrices1</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in Diagonal_Matrices2</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Data type of scale factor</td>
</tr>
<tr>
<td>Diagonal_Rang1</td>
<td>integer type</td>
<td>Used to dimension Diagonal_Matrices1</td>
</tr>
<tr>
<td>Diagonal_Rang2</td>
<td>integer type</td>
<td>Used to dimension Diagonal_Matrices2</td>
</tr>
<tr>
<td>Diagonal_Matrices1</td>
<td>array</td>
<td>An array of Elements1</td>
</tr>
<tr>
<td>Diagonal_Matrices2</td>
<td>array</td>
<td>An array of Elements2</td>
</tr>
</tbody>
</table>

3.3.6.2.9.10.2.8 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:

\[ a(i,j) := b(i,k) \ast c(k,j) \]

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating point type</td>
<td>Data type of elements in left input matrix</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating point type</td>
<td>Data type of elements in right input matrix</td>
</tr>
<tr>
<td>Output_Elements</td>
<td>floating point type</td>
<td>Data type of elements in output matrix</td>
</tr>
<tr>
<td>Left_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of left input matrix</td>
</tr>
<tr>
<td>Left_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of left input matrix</td>
</tr>
<tr>
<td>Right_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of right input matrix</td>
</tr>
<tr>
<td>Right_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of right input matrix</td>
</tr>
<tr>
<td>Output_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of output matrix</td>
</tr>
<tr>
<td>Output_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of output matrix</td>
</tr>
<tr>
<td>Left_Matrices</td>
<td>array type</td>
<td>Data type of left input matrix</td>
</tr>
<tr>
<td>Right_Matrices</td>
<td>array type</td>
<td>Data type of right input matrix</td>
</tr>
<tr>
<td>Output_Matrices</td>
<td>array type</td>
<td>Data type of output matrix</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| "*"  | 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: \([m \times n] \times [n \times p] := [m \times p]\)
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:
3.3.6.2.9.10.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the dimensions of the matrices are other than:</td>
</tr>
<tr>
<td></td>
<td>([m \times n] \times [n \times p] = [m \times p])</td>
</tr>
</tbody>
</table>

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 \(m \times n\) matrix by an \(n \times p\) matrix, returning an \(m \times p\) matrix.

The result of this operation is defined as follows:

\[ a(i,j) := b(i,k) \times c(k,j) \]

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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left_Matrices</td>
<td>In</td>
<td>( m \times n ) matrix</td>
</tr>
<tr>
<td>Right</td>
<td>Right_Matrices</td>
<td>In</td>
<td>( n \times p ) matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.10.10.1.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Output_Matrices</td>
<td>N/A</td>
<td>Result matrix</td>
</tr>
<tr>
<td>M_Answer</td>
<td>Output_Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of result matrix</td>
</tr>
<tr>
<td>M_Left</td>
<td>Left_Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of left input matrix</td>
</tr>
<tr>
<td>N_Left</td>
<td>Left_Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of left input matrix</td>
</tr>
<tr>
<td>N_Right</td>
<td>Right_Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of right input matrix</td>
</tr>
<tr>
<td>P_Answer</td>
<td>Output_Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of result matrix</td>
</tr>
<tr>
<td>P_Right</td>
<td>Right_Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of right input matrix</td>
</tr>
</tbody>
</table>

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:

\[
\text{function } \ast \text{ (Left : Left_Matrices; Right : Right_Matrices) return Output_Matrices is}
\]

-- 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;
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating point type</td>
<td>Data type of elements in left input matrix</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating point type</td>
<td>Data type of elements in right input matrix</td>
</tr>
<tr>
<td>Output_Elements</td>
<td>floating point type</td>
<td>Data type of elements in output matrix</td>
</tr>
<tr>
<td>Left_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of left input matrix</td>
</tr>
<tr>
<td>Left_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of left input matrix</td>
</tr>
<tr>
<td>Right_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of right input matrix</td>
</tr>
<tr>
<td>Right_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of right input matrix</td>
</tr>
<tr>
<td>Output_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of output matrix</td>
</tr>
<tr>
<td>Output_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of output matrix</td>
</tr>
<tr>
<td>Left_Matrices</td>
<td>array type</td>
<td>Data type of left input matrix</td>
</tr>
<tr>
<td>Right_Matrices</td>
<td>array type</td>
<td>Data type of right input matrix</td>
</tr>
<tr>
<td>Output_Matrices</td>
<td>array type</td>
<td>Data type of output matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation Left_Elements * Right_Elements := Output_Elements</td>
</tr>
<tr>
<td>&quot;+&quot;</td>
<td>function</td>
<td>Function defining the operation Output_Elements + Output_Elements := Output_Elements</td>
</tr>
</tbody>
</table>

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 \( m \times n \) matrix by an \( n \times 1 \) vector producing an \( m \times 1 \) 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:

\[
a(i) := b(i,j) \times c(j)
\]

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 : \( m \times n \) array Input_Vectors : \( n \times 1 \) array Output_Vectors : \( m \times 1 \) 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:
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>★</td>
<td>function</td>
<td>Function defining the operation</td>
</tr>
<tr>
<td>★+</td>
<td>function</td>
<td>Function defining the operation</td>
</tr>
</tbody>
</table>

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 l] = [m x l]
if NOT (Input_Matrices'LENGTH(2) = Input_Vectors'LENGTH and
Input_Matrices'LENGTH(1) = Output_Vectors'LENGTH) then

--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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.11.8 LIMITATIONS

The following table describes the exceptions raised by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the length if an operation other than the following is attempted: [m x n] x [n x 1] := [m x 1]</td>
</tr>
</tbody>
</table>

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:

\[ a(i) := b(i,j) \times c(j) \]

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Input Matrices</td>
<td>In</td>
<td>Matrix to be used as the multiplicand</td>
</tr>
<tr>
<td>Vector</td>
<td>Input Vectors</td>
<td>In</td>
<td>Vector to be used as the multiplier</td>
</tr>
</tbody>
</table>

3.3.6.2.9.11.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Output_Vectors</td>
<td>Result of performing the matrix-vector multiplication</td>
</tr>
<tr>
<td>M_Answer</td>
<td>Output_Vector_Indices</td>
<td>Index into result vector</td>
</tr>
<tr>
<td>M_Matrix</td>
<td>Row_Indices</td>
<td>Index into input matrix</td>
</tr>
<tr>
<td>N_Matrix</td>
<td>Col_Indices</td>
<td>Index into input matrix</td>
</tr>
<tr>
<td>N_Vector</td>
<td>Input_Vector_Indices</td>
<td>Index into input vector</td>
</tr>
</tbody>
</table>

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:

```cpp
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);
  exit M_Loop when M_Answer = Output_Vector_Indices'LAST;
  M_Answer := Output_Vector_Indices'SUCC(M_Answer);

end function "*
```
end loop N_Loop;

exit M Loop when M_Matrix = Row_Indices'LAST;
M_Matrix := Row_Indices'SUC(M_Matrix);
M_Answer := Output_Vector_Indices'SUC(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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix_Elements</td>
<td>floating</td>
<td>Type of elements in the input matrix</td>
</tr>
<tr>
<td>Input_Vector_Elements</td>
<td>floating</td>
<td>Type of elements in the input vector</td>
</tr>
<tr>
<td>Output_Vector_Elements</td>
<td>floating</td>
<td>Type of elements in the output vector</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to dimension second dimension of input matrix</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to dimension first dimension of input matrix</td>
</tr>
<tr>
<td>Input_Vector_Indices</td>
<td>discrete</td>
<td>Used to dimension input vector</td>
</tr>
<tr>
<td>Output_Vector_Indices</td>
<td>discrete</td>
<td>Used to dimension output vector</td>
</tr>
<tr>
<td>Input_Matrices</td>
<td>array</td>
<td>Data type of input matrix</td>
</tr>
<tr>
<td>Input_Vectors</td>
<td>array</td>
<td>Data type of input vector</td>
</tr>
<tr>
<td>Output_Vectors</td>
<td>array</td>
<td>Data type of output vector</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| "*"  | 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:

\[
a(i,j) := b(i) \times c(j)
\]

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_VectorTranspose_Multiply_Unrestricted package:

Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Vector_Elements</td>
<td>floating point</td>
<td>Data type of elements in left input vector</td>
</tr>
<tr>
<td>Right_Vector_Elements</td>
<td>floating point</td>
<td>Data type of elements in right input vector</td>
</tr>
<tr>
<td>Matrix_Elements</td>
<td>floating point</td>
<td>Data type of elements in output matrix</td>
</tr>
<tr>
<td>Left_Vector_Indices</td>
<td>discrete type</td>
<td>Used to dimension left input vector</td>
</tr>
<tr>
<td>Right_Vector_Indices</td>
<td>discrete type</td>
<td>Used to dimension right input vector</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of output matrix</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of output matrix</td>
</tr>
<tr>
<td>Left_Vectors</td>
<td>array</td>
<td>Data type of left input vector</td>
</tr>
<tr>
<td>Right_Vectors</td>
<td>array</td>
<td>Data type of right input vector</td>
</tr>
<tr>
<td>Matrices</td>
<td>array</td>
<td>Data type of output matrix</td>
</tr>
</tbody>
</table>

Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Operator defining the multiplication operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Left_Vector_Elements * Right_Vector_Elements :=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Matrix Elements</td>
</tr>
</tbody>
</table>

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

-- make sure dimensions are compatible; must have the following conditions:
-- attempted operation is \([m \times 1] \times [1 \times n] = [m \times n]\)
if NOT (Left_Vectors'LENGTH = Matrices'LENGTH(1) and Right_Vectors'LENGTH = Matrices'LENGTH(2)) then
  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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.12.8 LIMITATIONS

The following table describes the exceptions raised by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if an attempt is made to put the result of [ \begin{bmatrix} m \times 1 \end{bmatrix} \text{vector} \times \begin{bmatrix} 1 \times n \end{bmatrix} \text{vector} ] into other than a [ \begin{bmatrix} m \times n \end{bmatrix} \text{matrix} ]</td>
</tr>
</tbody>
</table>

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:

\[ a(i,j) := b(i) \times c(j) \]

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left_Vectors</td>
<td>In</td>
<td>m x 1 vector</td>
</tr>
<tr>
<td>Right</td>
<td>Right_Vectors</td>
<td>In</td>
<td>1 x n vector</td>
</tr>
</tbody>
</table>

3.3.6.2.9.12.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result .rix</td>
</tr>
<tr>
<td>M_Answer</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of output matrix</td>
</tr>
<tr>
<td>M_Left</td>
<td>Left_Vector_Indices</td>
<td>N/A</td>
<td>Index into left input vector</td>
</tr>
<tr>
<td>N_Answer</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of output matrix</td>
</tr>
<tr>
<td>N_Right</td>
<td>Right_Vector_Indices</td>
<td>N/A</td>
<td>Index into right input vector</td>
</tr>
</tbody>
</table>

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:

```plaintext
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);

d  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_VectorTranspose_Multiply_Unrestricted package:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Vector_Elements</td>
<td>floating point type</td>
<td>Data type of elements in left input vector</td>
</tr>
<tr>
<td>Right_Vector_Elements</td>
<td>floating point type</td>
<td>Data type of elements in right input vector</td>
</tr>
<tr>
<td>Matrix_Elements</td>
<td>floating point type</td>
<td>Data type of elements in output matrix</td>
</tr>
<tr>
<td>Left_Vector_Indices</td>
<td>discrete type</td>
<td>Used to dimension left input vector</td>
</tr>
<tr>
<td>Right_Vector_Indices</td>
<td>discrete type</td>
<td>Used to dimension right input vector</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of output matrix</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of output matrix</td>
</tr>
<tr>
<td>Left_Vectors</td>
<td>array</td>
<td>Data type of left input vector</td>
</tr>
<tr>
<td>Right_Vectors</td>
<td>array</td>
<td>Data type of right input vector</td>
</tr>
<tr>
<td>Matrices</td>
<td>array</td>
<td>Data type of output matrix</td>
</tr>
</tbody>
</table>

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_VectorTranspose_Multiply_Unrestricted package:
### 3.3.6.2.9.13 MATRIX MATRIX TRANSPOSE MULTIPLY UNRESTRICTED PACKAGE DESIGN (CATALOG #P445-O)

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:

\[ a(i,j) := b(i,k) \times c(j,k) \]

This package expects the instantiated arrays to have been dimensioned as follows:

- **Left Matrices**: \( m \times n \) matrix
- **Right Matrices**: \( p \times n \) matrix
- **Output Matrices**: \( m \times p \) 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating point type</td>
<td>Type of elements in left input matrix</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating point type</td>
<td>Type of elements in right input matrix</td>
</tr>
<tr>
<td>Output_Elements</td>
<td>floating point type</td>
<td>Type of elements in output matrix</td>
</tr>
<tr>
<td>Left_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of left input matrix</td>
</tr>
<tr>
<td>Left_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of left input matrix</td>
</tr>
<tr>
<td>Right_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of right input matrix</td>
</tr>
<tr>
<td>Right_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of right input matrix</td>
</tr>
<tr>
<td>Output_Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of output matrix</td>
</tr>
<tr>
<td>Output_Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of output matrix</td>
</tr>
<tr>
<td>Left_Matrices</td>
<td>array</td>
<td>Data type of left input matrix</td>
</tr>
<tr>
<td>Right_Matrices</td>
<td>array</td>
<td>Data type of right input matrix</td>
</tr>
<tr>
<td>Output_Matrices</td>
<td>array</td>
<td>Data type of output matrix</td>
</tr>
</tbody>
</table>

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Operator used to define the operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Left_Elements * Right_Elements := Output_Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.13.8 LIMITATIONS

The following table describes the exceptions raised by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>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 \times n] \times [p \times n] := [m \times p])</td>
</tr>
</tbody>
</table>

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 \(m \times n\) matrix by the transpose of a \(p \times n\) matrix, returning the resultant \(m \times p\) matrix.

The results of this operation are defined as follows:

\[a(i, j) := b(i, k) \times c(j, k)\]

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left Matrices</td>
<td>In</td>
<td>Matrix to be used as the multiplicand</td>
</tr>
<tr>
<td>Right</td>
<td>Right Matrices</td>
<td>In</td>
<td>Matrix whose transpose is to be used as the multiplier</td>
</tr>
</tbody>
</table>

3.3.6.2.9.13.10.1.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Output Matrices</td>
<td>N/A</td>
<td>Result matrix being calculated</td>
</tr>
<tr>
<td>M_Answer</td>
<td>Output_Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of result matrix</td>
</tr>
<tr>
<td>M_Left</td>
<td>Left_Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of left input matrix</td>
</tr>
<tr>
<td>N_Left</td>
<td>Left_Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of left input matrix</td>
</tr>
<tr>
<td>N_Right</td>
<td>Right_Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of right input matrix</td>
</tr>
<tr>
<td>P_Answer</td>
<td>Output_Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of result matrix</td>
</tr>
<tr>
<td>P_Right</td>
<td>Right_Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of right input matrix</td>
</tr>
</tbody>
</table>

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;
CAMP Software Detailed Design Document

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'SUC(S_Right);
        N_Right := Right_Col_Indices'SUC(N_Right);
      end loop N_Loop;
      exit P_Loop when P_Answer = Output_Col_Indices'LAST;
      P_Answer := Output_Col_Indices'SUC(P_Answer);
      P_Right := Right_Row_Indices'SUC(P_Right);
    end loop P_Loop;
    exit M_Loop when M_Answer = Output_Row_Indices'LAST;
    M_Answer := Output_Row_Indices'SUC(M_Answer);
    M_Left := Left_Row_Indices'SUC(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:
### 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating</td>
<td>Type of elements in left input matrix</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating</td>
<td>Type of elements in right input matrix</td>
</tr>
<tr>
<td>Output_Elements</td>
<td>floating</td>
<td>Type of elements in output matrix</td>
</tr>
<tr>
<td>Left_Col_Indices</td>
<td>discrete</td>
<td>Used to dimension second dimension of left input matrix</td>
</tr>
<tr>
<td>Left_Row_Indices</td>
<td>discrete</td>
<td>Used to dimension first dimension of left input matrix</td>
</tr>
<tr>
<td>Right_Col_Indices</td>
<td>discrete</td>
<td>Used to dimension second dimension of right input matrix</td>
</tr>
<tr>
<td>Right_Row_Indices</td>
<td>discrete</td>
<td>Used to dimension first dimension of right input matrix</td>
</tr>
<tr>
<td>Output_Col_Indices</td>
<td>discrete</td>
<td>Used to dimension second dimension of output matrix</td>
</tr>
<tr>
<td>Output_Row_Indices</td>
<td>discrete</td>
<td>Used to dimension first dimension of output matrix</td>
</tr>
<tr>
<td>Left_Matrices</td>
<td>array</td>
<td>Data type of left input matrix</td>
</tr>
<tr>
<td>Right_Matrices</td>
<td>array</td>
<td>Data type of right input matrix</td>
</tr>
<tr>
<td>Output_Matrices</td>
<td>array</td>
<td>Data type of output matrix</td>
</tr>
</tbody>
</table>

#### 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 "^n-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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating point type</td>
<td>Type of elements in left input vector</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating point type</td>
<td>Type of elements in right input vector</td>
</tr>
<tr>
<td>Result_Elements</td>
<td>floating point type</td>
<td>Data type of result of dot product</td>
</tr>
<tr>
<td>Left_Indices</td>
<td>discrete</td>
<td>Used to dimension Left_Vectors</td>
</tr>
<tr>
<td>Right_Indices</td>
<td>discrete</td>
<td>Used to dimension Right_Vectors</td>
</tr>
<tr>
<td>Left_Vectors</td>
<td>array</td>
<td>Data type of left input vector</td>
</tr>
<tr>
<td>Right_Vectors</td>
<td>array</td>
<td>Data type of right input vector</td>
</tr>
</tbody>
</table>

Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Multiplication operator defining the operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Left_Elements * Right_Elements := Result_Elements</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>
3.3.6.2.9.14.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the lengths of the two input vectors is not the same</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left_Vectors</td>
<td>in</td>
<td>First vector in a dot product operation</td>
</tr>
<tr>
<td>Right</td>
<td>Right_Vectors</td>
<td>in</td>
<td>Second vector in a dot product operation</td>
</tr>
</tbody>
</table>

3.3.6.2.9.14.10.1.4 LOCAL DATA

Data objects:

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

```pl
function Dot_Product (Left : Left_Vectors;
                     Right : Right_Vectors) return Result_Elements is

-- ---declaration section---
-- ---end declaration section---

    Answer    : Result_Elements;
    L_Index   : Left_Indices;
    R_Index   : Right_Indices;

-- ---begin function Dot_Product---
-- ---end 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating point type</td>
<td>Type of elements in left input vector</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating point type</td>
<td>Type of elements in right input vector</td>
</tr>
<tr>
<td>Result_Elements</td>
<td>floating point type</td>
<td>Data type of result of dot product</td>
</tr>
<tr>
<td>Left_Indices</td>
<td>discrete</td>
<td>Used to dimension Left_Vectors</td>
</tr>
<tr>
<td>Right_Indices</td>
<td>discrete</td>
<td>Used to dimension Right_Vectors</td>
</tr>
<tr>
<td>Left_Vectors</td>
<td>array</td>
<td>Data type of left input vector</td>
</tr>
<tr>
<td>Right_Vectors</td>
<td>array</td>
<td>Data type of right input vector</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Type of elements in input and output arrays</td>
</tr>
<tr>
<td>Diagonal_Range</td>
<td>integer</td>
<td>Used to dimension Diagonal_Matrices</td>
</tr>
<tr>
<td>Full_Input_Col_Indices</td>
<td>discrete</td>
<td>Used to dimension Full_Input_matrices</td>
</tr>
<tr>
<td>Full_Input_Row_Indices</td>
<td>discrete</td>
<td>Used to dimension Full_Input_matrices</td>
</tr>
<tr>
<td>Full_Ouput_Col_Indices</td>
<td>discrete</td>
<td>Used to dimension Full_Output_matrices</td>
</tr>
<tr>
<td>Full_Output_Row_Indices</td>
<td>discrete</td>
<td>Used to dimension Full_Output_matrices</td>
</tr>
<tr>
<td>Diagonal_Matrices</td>
<td>array</td>
<td>Data type of diagonal input matrix</td>
</tr>
<tr>
<td>Full_Input_Matrix</td>
<td>array</td>
<td>Data type of full input matrix</td>
</tr>
<tr>
<td>Full_Output_Matrix</td>
<td>array</td>
<td>Data type of full output matrix</td>
</tr>
</tbody>
</table>

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:

```pascal
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.15.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the lengths of the matrix indices are not equal to each other and other the length of the diagonal matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMatrix</td>
<td>Diagonal_Matrices</td>
<td>In</td>
<td>Input diagonal matrix</td>
</tr>
<tr>
<td>FMatrix</td>
<td>Full_Input_Matrices</td>
<td>In</td>
<td>Input full matrix to be added to the diagonal matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.15.10.1.4  LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Full Output Matrices</td>
<td>N/A</td>
<td>Resultant matrix</td>
</tr>
<tr>
<td>A_Col_Index</td>
<td>Full_Output_Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of Answer matrix</td>
</tr>
<tr>
<td>A_Col_Marker</td>
<td>Full_Output_Col_Indices</td>
<td>N/A</td>
<td>Marks a column in Answer matrix which contains the diagonal element in row A_Row_Index</td>
</tr>
<tr>
<td>A_Row_Index</td>
<td>Full_Output_Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of Answer matrix</td>
</tr>
<tr>
<td>D_Index</td>
<td>Diagonal_Range</td>
<td>N/A</td>
<td>Index into diagonal matrix</td>
</tr>
<tr>
<td>F_Col_Index</td>
<td>Full_Input_Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of F_Matrix</td>
</tr>
<tr>
<td>F_Row_Index</td>
<td>Full_Input_Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of F_Matrix</td>
</tr>
</tbody>
</table>

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:

```plaintext
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;
  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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dot_Product</td>
<td>R063</td>
</tr>
<tr>
<td>Vector_Length</td>
<td>R104</td>
</tr>
<tr>
<td>&quot;-&quot;</td>
<td>R061</td>
</tr>
<tr>
<td>&quot;-&quot;</td>
<td>R062</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements to be contained in vector type defined by this package</td>
</tr>
<tr>
<td>Vector_Elements_Squared</td>
<td>floating point type</td>
<td>Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

Subprograms:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared</td>
</tr>
<tr>
<td>SqRt</td>
<td>function</td>
<td>Square root function taking an object of type Vector_Elements_Squared and returning an object of type Vector_Elements</td>
</tr>
</tbody>
</table>

3.3.6.2.9.16.4 LOCAL DATA

Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Constrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>
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:

```haskell
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 : VECTOR) 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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Vectors</td>
<td>In</td>
<td>One of the vectors to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Vectors</td>
<td>In</td>
<td>Second vector to be added</td>
</tr>
</tbody>
</table>

3.3.6.2.9.16.10.1.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vectors</td>
<td>Vector being calculated and returned</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements to be contained in vector type defined by this package</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Constrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Vectors</td>
<td>In</td>
<td>Vector to act as the minuend</td>
</tr>
<tr>
<td>Right</td>
<td>Vectors</td>
<td>In</td>
<td>Vector to act as the subtrahend</td>
</tr>
</tbody>
</table>

3.3.6.2.9.16.10.2.4  LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vectors</td>
<td>Vector being calculated and returned</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements to be contained in vector type defined by this package</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Constrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>

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:

\[ a := \text{Sqrt} \left( \sum b(i)^2 \right) \]

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Vectors</td>
<td>In</td>
<td>Vector for which a length is desired</td>
</tr>
</tbody>
</table>

3.3.6.2.9.16.10.3.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temp</td>
<td>Vector_Elements_Squared</td>
<td>Used for intermediate calculations</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements to be contained in vector type defined by this package</td>
</tr>
<tr>
<td>Vector_Elements_Squared</td>
<td>floating point type</td>
<td>Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Constrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>
Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared</td>
</tr>
<tr>
<td>SqRt</td>
<td>function</td>
<td>Square root function taking an object of type Vector_Elements_Squared and returning an object of type Vector_Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Vectors</td>
<td>In</td>
<td>First vector to be used in the dot product operation</td>
</tr>
<tr>
<td>Right</td>
<td>Vectors</td>
<td>In</td>
<td>Second vector to be used in the dot product operation</td>
</tr>
</tbody>
</table>
3.3.6.2.9.16.10.4.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vector_Elements_Squared</td>
<td>N/A</td>
<td>Result of dot product operation</td>
</tr>
</tbody>
</table>

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:

```plaintext
function Dot_Product (Left : Vectors;
                      Right : Vectors) return Vector
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements to be contained in vector type defined by this package</td>
</tr>
<tr>
<td>Vector_Elements_Squared</td>
<td>floating point type</td>
<td>Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension exported Vectors type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>N/A</td>
<td>Constrained, one-dimensional array of elements</td>
</tr>
</tbody>
</table>

Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>function</td>
<td>Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;+&quot; (matrices + matrices)</td>
<td>R079</td>
</tr>
<tr>
<td>&quot;-&quot; (matrices - matrices)</td>
<td>R080</td>
</tr>
<tr>
<td>&quot;+&quot; (matrices + elements)</td>
<td>R075</td>
</tr>
<tr>
<td>&quot;-&quot; (matrices - elements)</td>
<td>R076</td>
</tr>
<tr>
<td>Set_to_Identity_Matrix</td>
<td>R155</td>
</tr>
<tr>
<td>Set_to_Zero_Matrix</td>
<td>R156</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

3.3.6.2.9.17.4 LOCAL DATA

Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
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:
3.3.6.2.9.17.10.1.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Result of adding the two input matrices</td>
</tr>
</tbody>
</table>

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:

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

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;

  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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td></td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to act as the minuend</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be used as the subtrahend</td>
</tr>
</tbody>
</table>

3.3.6.2.9.17.10.2.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Result of adding the two input matrices</td>
</tr>
</tbody>
</table>

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:

```pascal
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Addend</td>
<td>Elements</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.17.10.3.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Scaled matrix</td>
</tr>
</tbody>
</table>

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:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
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:


<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Subtrahend</td>
<td>Elements</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.17.10.4.4 LOCAL DATA

Data objects:

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


<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>Scaled matrix</td>
</tr>
</tbody>
</table>

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:

```vhdl
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>
The following table summarizes the types required by this part and defined in
the package specification of Matrix_Operations_Constrained:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Matrix to be made into an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.17.10.5.4 LOCAL DATA

Data objects:
The following describes the data objects maintained local to this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>Index into second dimension of matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>Index into first dimension of matrix</td>
</tr>
</tbody>
</table>

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:

```plaintext
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));
    end if;
    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;
      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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Used to define type of elements in matrix</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td>defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has</td>
</tr>
<tr>
<td></td>
<td>dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.17.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Matrix to be zeroed out</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Used to define type of elements in matrix defined by this package</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to define second dimension of exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to define first dimension of exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.4 LOCAL DATA

Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix being made into an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>Index into second dimension of input matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>Index into first dimension of input matrix</td>
</tr>
</tbody>
</table>

### 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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be zeroed out</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to which is added an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.10.3.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding an identity matrix to the input matrix</td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index</td>
</tr>
</tbody>
</table>

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:

```vhdl
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>
The following types are defined in the package specification for Dynamically_Sparse_Matrix_Operations_Constrained:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Square matrix to be subtracted from an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.10.4.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of subtracting input matrix from an identity matrix</td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index</td>
</tr>
</tbody>
</table>

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:

```vhdl
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;

3.3.2.9.18.10.4.7 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>
The following types are defined in the package specification for Dynamically-Sparse_Matrix_Operations_Constrained:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

Exceptions:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>Sparse matrix to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Sparse matrix to be added</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.10.5.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding two input matrices</td>
</tr>
</tbody>
</table>

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:

```pascal
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
```
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
3.3.6.2.9.18.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if both matrices are not m x n matrices</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>Sparse matrix to be treated as the minuend</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Sparse matrix to be treated as the subtrahend</td>
</tr>
</tbody>
</table>

3.3.6.2.9.18.10.6.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of subtracting two input matrices</td>
</tr>
</tbody>
</table>
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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_error</td>
<td>Raised by a routine when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

3.3.6.2.9.19.4 LOCAL DATA

Data types:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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_Matrlx_Operatlons_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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine or package when input received has dimensions incompatible for the type of operation to be performed</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New_Value</td>
<td>Elements</td>
<td>In</td>
<td>New value to be placed in the matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>In</td>
<td>Row in which the value belongs</td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>In</td>
<td>Column in which the value belongs</td>
</tr>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>In/Out</td>
<td>Matrix being updated</td>
</tr>
</tbody>
</table>

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:

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

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

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
or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>point type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Matrix to be made into an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.19.10.2.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of matrix</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of matrix</td>
</tr>
</tbody>
</table>

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 or 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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices</td>
<td>Out</td>
<td>Matrix to be zeroed out</td>
</tr>
</tbody>
</table>
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:

```plaintext
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 or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>point type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be added to an identity matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.9.19.10.4.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding identity matrix to input matrix</td>
</tr>
<tr>
<td>Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Index into second dimension of matrices</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of matrices</td>
</tr>
</tbody>
</table>

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:

\[
\text{function Add_to_Identity (Input : Matrices) return Matrices is}
\]

\[
\begin{aligned}
\text{-- \textit{--declaration section}} \\
\text{-- \textit{--declaration section}} \\
\text{Answer : Matrices;} \\
\text{Col : Col_Indices;} \\
\text{Row : Row_Indices;}
\end{aligned}
\]

\[
\begin{aligned}
\text{-- \textit{--begin function Add_to_Identity}} \\
\text{-- \textit{--begin function Add_to_Identity}} \\
\begin{aligned}
\text{begin} \\
\text{Answer := Input;} \\
\text{Col := Col_Indices'FIRST;} \\
\text{Row := Row_Indices'FIRST;} \\
\text{Access_Diagonal_Elements:} \\
\text{\quad loop} \\
\text{\quad \quad Answer(Row,Col) := Answer(Row,Col) + 1.0;} \\
\text{\quad \quad exit when Row = Row_Indices'LAST;} \\
\text{\quad \quad Col := Col_Indices'SUCC(Col);} \\
\text{\quad \quad Row := Row_Indices'SUCC(Row);} \\
\text{\quad end loop Access_Diagonal_Elements;} \\
\text{\quad return Answer;} \\
\text{end Add_to_Identity;}
\end{aligned}
\end{aligned}
\]

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 or 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:
### Elements

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>point type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

#### LIMITATIONS

None.

#### SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P403-0)

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

#### REQUIREMENTS ALLOCATION

See top header.

#### LOCAL ENTITIES DESIGN

None.

#### INPUT/OUTPUT

**FORMAL PARAMETERS:**

The following table describes this part's formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be subtracted from an identity matrix</td>
</tr>
</tbody>
</table>
3.3.6.2.9.19.10.5.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding input matrices</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into matrix</td>
</tr>
<tr>
<td>S_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>&quot;Symmetric&quot; column index into matrix</td>
</tr>
<tr>
<td>S_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>&quot;Symmetric&quot; row index into matrix; i.e., A(rov, col) := A(s_row, s_col)</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>
The following table summarizes the types required by this part and defined in
the package specification of Symmetric_Full_Storage_Matrix_Operations_Constrained:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

3.3.6.2.9.19.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if input matrix is not a square matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>First matrix to be added</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Second matrix to be added</td>
</tr>
</tbody>
</table>
3.3.6.2.9.19.10.6.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding two input matrices</td>
</tr>
<tr>
<td>Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Index into first dimension of matrix</td>
</tr>
<tr>
<td>S_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Used to keep track of column containing diagonal element for the current row</td>
</tr>
</tbody>
</table>
| 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., \( A(\text{row},\text{col}) = A(\text{s\_row},\text{s\_col}) \)

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating</td>
<td>Data type of elements in exported matrix</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td></td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>
The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations-Constrained:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td>N/A</td>
<td>Constrained, two-dimensional array of Elements</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be subtracted from</td>
</tr>
<tr>
<td>Right</td>
<td>Matrices</td>
<td>In</td>
<td>Matrix to be used as the subtrahend</td>
</tr>
</tbody>
</table>

3.3.6.2.9.19.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result of adding two input matrices</td>
</tr>
<tr>
<td>Row</td>
<td>Row Indices</td>
<td>N/A</td>
<td>Index into first dimension of matrix</td>
</tr>
<tr>
<td>S_Col</td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Used to keep track of column containing diagonal element for the current row</td>
</tr>
<tr>
<td>S_Row</td>
<td>Row_Indices</td>
<td>N/A</td>
<td>When used with S_Col, marks the symmetric counterpart to the element being</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>referenced in the bottom half of the array; i.e., A(row,col) = A(s_row,s_col)</td>
</tr>
</tbody>
</table>

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:

```pascal
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
```

```pascal```
-- 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
or 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Data type of elements in exported matrix type</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension exported matrix type</td>
</tr>
</tbody>
</table>

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

None.

3.3.6.2.9.20 VECTOR_SCALAR_OPERATIONS_CONSTRANGED 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>RO65</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>RO66</td>
</tr>
</tbody>
</table>

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:
### Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Used to define the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elements1 := Elements2 * Scalars</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>function</td>
<td>Used to define the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elements2 := Elements1 / Scalars</td>
</tr>
</tbody>
</table>

### 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:

```pascal
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Vectors2</td>
<td>In</td>
<td>Vector to be scaled</td>
</tr>
<tr>
<td>Multiplier</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.20.10.1.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Vectors1</td>
<td>Scaled vector</td>
</tr>
</tbody>
</table>
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:

```pascal
function "*" (Vector : Vectors2; Multiplier : Scalars) return Vectors1 is

  Answer : Vectors1;

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:
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>/</em></td>
<td>function</td>
<td>Used to define the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elements1 := Elements2 * Scalars</td>
</tr>
</tbody>
</table>

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:
3.3.6.2.9.20.10.2.4 LOCAL DATA

Data objects:

The following describes the local data maintained by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Vectors1</td>
<td>Vector to be scaled</td>
</tr>
<tr>
<td>Divisor</td>
<td>Scalars</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

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:

```haskell
function "/" (Vector : Vectors1; Divisor : Scalars) return Vectors2 is

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in a vector; $\text{Elements1} := \text{Elements2} \ast \text{Scalars}$</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in a vector; $\text{Elements2} := \text{Elements1} / \text{Scalars}$</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Type of value to be used for multiplying and dividing</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension vectors</td>
</tr>
<tr>
<td>Vectors1</td>
<td>array</td>
<td>An array of Elements1</td>
</tr>
<tr>
<td>Vectors2</td>
<td>array</td>
<td>An array of Elements2</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;/&quot;</td>
<td>function</td>
<td>Used to define the operation [ \text{Elements2} := \text{Elements1} / \text{Scalars} ]</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>R073</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>R074</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Data type of objects to be used as multipliers and divisors</td>
</tr>
<tr>
<td>Col</td>
<td>discrete type</td>
<td>Used to dimension second dimension of matrices</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of matrices</td>
</tr>
<tr>
<td>Matrices1</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements1</td>
</tr>
<tr>
<td>Matrices2</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements2</td>
</tr>
</tbody>
</table>

Subprograms:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function to define the operation Elements1 * Scalars := Elements2</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>function</td>
<td>Function to define the operation Elements2 / Scalars := Elements1</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices1</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Multiplier</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.21.10.1.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices2</td>
<td>Scaled matrix</td>
</tr>
</tbody>
</table>

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:

```pl
function "*" (Matrix : Matrices1; Multiplier : Scalars) return Matrices2 is

Answer : Matrices2;

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;
extend "*";

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Data type of objects to be used as multipliers and divisors</td>
</tr>
<tr>
<td>Col</td>
<td>discrete type</td>
<td>Used to dimension second dimension of matrices</td>
</tr>
<tr>
<td>Indices</td>
<td>type</td>
<td>Used to dimension first dimension of matrices</td>
</tr>
<tr>
<td>Row</td>
<td>discrete type</td>
<td></td>
</tr>
<tr>
<td>Indices</td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>Matrices1</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements1</td>
</tr>
<tr>
<td>Matrices2</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements2</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function to define the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elements1 * Scalars := Elements2</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrices2</td>
<td>In</td>
<td>Matrix to be scaled</td>
</tr>
<tr>
<td>Divisor</td>
<td>Scalars</td>
<td>In</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>

3.3.6.2.9.21.10.2.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices1</td>
<td>Scaled matrix</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements1</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Elements2</td>
<td>floating point type</td>
<td>Type of elements in an array</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Data type of objects to be used as multipliers and divisors</td>
</tr>
<tr>
<td>Col</td>
<td>discrete type</td>
<td>Used to dimension second dimension of matrices</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of matrices</td>
</tr>
<tr>
<td>Matrices1</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements1</td>
</tr>
<tr>
<td>Matrices2</td>
<td>array</td>
<td>Two dimensional matrix with elements of type Elements2</td>
</tr>
</tbody>
</table>

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 $1 \times m$ vector by an $m \times n$ matrix producing a $1 \times n$ 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:
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Vector Elements</td>
<td>floating point type</td>
<td>Type of elements in the input vector</td>
</tr>
<tr>
<td>Matrix Elements</td>
<td>floating point type</td>
<td>Type of elements in the input matrix</td>
</tr>
<tr>
<td>Output Vector Elements</td>
<td>floating point type</td>
<td>Type of elements in the output vector</td>
</tr>
<tr>
<td>Col_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of input matrix</td>
</tr>
<tr>
<td>Row_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of input matrix</td>
</tr>
<tr>
<td>Input Vector Indices</td>
<td>discrete type</td>
<td>Used to dimension input vector</td>
</tr>
<tr>
<td>Output Vector Indices</td>
<td>discrete type</td>
<td>Used to dimension output vector</td>
</tr>
<tr>
<td>Input Matrices</td>
<td>array</td>
<td>Data type of input matrix</td>
</tr>
<tr>
<td>Input Vectors</td>
<td>array</td>
<td>Data type of input vector</td>
</tr>
<tr>
<td>Output Vectors</td>
<td>array</td>
<td>Data type of output vector</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation</td>
</tr>
</tbody>
</table>
|       |              | \[\text{Input}_V \times \text{Elements} \times \text{Matrix}_\text{Elements} := \]
|       |              | \[\text{Output}_\text{Vector}_\text{Elements} \]
| "+"   | function     | Function defining the operation                  |
|       |              | \[\text{Output}_\text{Vector}_\text{Elements} + \]
|       |              | \[\text{Output}_\text{Vector}_\text{Elements} := \]
|       |              | \[\text{Output}_\text{Vector}_\text{Elements} \]

FORMAL PARAMETERS:

The following table describes the formal parameters for the "*" unit contained in this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Input_Vectors</td>
<td>1xM vector to be used in the calculation</td>
</tr>
<tr>
<td>Matrix</td>
<td>Input_Matrices</td>
<td>MxN matrix to be used in the calculation</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Output_Vectors</td>
<td>N/A</td>
<td>Result vector being calculated</td>
</tr>
<tr>
<td>M</td>
<td>Input_Vector_Indices</td>
<td>N/A</td>
<td>Index into the 1 x m input vector</td>
</tr>
<tr>
<td>N</td>
<td>Output_Vector_Indices</td>
<td>N/A</td>
<td>Index into the n x 1 output vector</td>
</tr>
<tr>
<td></td>
<td>Col_Indices</td>
<td>N/A</td>
<td>Column index into the m x n input matrix</td>
</tr>
<tr>
<td></td>
<td>Row_Indices</td>
<td>N/A</td>
<td>Row index into the m x n input matrix</td>
</tr>
</tbody>
</table>
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
-- --begin function "*"

begin

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 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: [lxm] * [mxn] => [lxn]

if Input_Vectors'LENGTH /= Input_Matrices'LENGTH(1) or
Input_Matrices'LENGTH(2) /= Output_Vectors'LENGTH then
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised by a routine or package when input received has dimensions incompatible with the type of operation to be performed</td>
</tr>
</tbody>
</table>

3.3.6.2.9.22.8 LIMITATIONS

The following table describes the exceptions raised by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>When/Why Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension_Error</td>
<td>Raised if the sizes of the data objects are incompatible for the multiplication operation</td>
</tr>
</tbody>
</table>
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 \((m \times n)\) and a square \((n \times n)\) matrix, yielding a square matrix. The first multiply \((A \times B)\) is constrained and the second \((A \times B^\top)\) is restricted.

The following table lists the catalog numbers for subunits contained in this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Catalog #</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABA_Transpose</td>
<td>P1067-0</td>
</tr>
</tbody>
</table>

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

GENERAL PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_Elements</td>
<td>floating point</td>
<td>Type of element in the dynamically sparse input matrix</td>
</tr>
<tr>
<td>B_Elements</td>
<td>floating point</td>
<td>Type of element in the square input matrix</td>
</tr>
<tr>
<td>C_Elements</td>
<td>floating point</td>
<td>Type of element in the output vector</td>
</tr>
<tr>
<td>M_Indices</td>
<td>discrete type</td>
<td>Used to dimension the 1st dimension of the sparse input matrix</td>
</tr>
<tr>
<td>N_Indices</td>
<td>discrete type</td>
<td>Used to dimension the 2nd dimension of the sparse matrix and both dimensions of the square matrix</td>
</tr>
<tr>
<td>A_Matrices</td>
<td>array</td>
<td>Data type of the dynamically sparse input matrix</td>
</tr>
<tr>
<td>B_Matrices</td>
<td>array</td>
<td>Data type of the square input matrix</td>
</tr>
<tr>
<td>C_Matrices</td>
<td>array</td>
<td>Data type of the output matrix</td>
</tr>
</tbody>
</table>

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation A_Elements * B_Elements := C_Elements</td>
</tr>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation C_Elements * A_Elements := C_Elements</td>
</tr>
</tbody>
</table>

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;
    end function;
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
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix_Multiply_Restricted</td>
<td>generic function</td>
<td>GVMA</td>
<td>Used to multiply the sparse matrix by the square matrix</td>
</tr>
<tr>
<td>Matrix_Transpose_Multiply</td>
<td>generic function</td>
<td>GVMA</td>
<td>Used to multiply the product of the first operation by the transpose of the sparse matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Catalog _#</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABA_Transpose</td>
<td>P1069-0</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiply_VM</td>
<td>function</td>
<td>Function defining the operation Vector_Elements * Matrix_Elements := Vector_Elements</td>
</tr>
<tr>
<td>Multiply_VV</td>
<td>function</td>
<td>Function defining the operation Vector_Elements * Vector_Elements := Scalars</td>
</tr>
<tr>
<td>Vector_Vector_Multiply</td>
<td>function</td>
<td>Function defining a vector vector multiply (dot product) Vectors * Vectors := Scalars Instanitation of GVMA.Dot_Product_Operations_Restricted</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating</td>
<td>Type of element in the input vector.</td>
</tr>
<tr>
<td>Matrix_Elements</td>
<td>floating</td>
<td>Type of element in the square input matrix.</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating</td>
<td>Type of element in the output scalar</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete</td>
<td>Used to dimension the input vector and both</td>
</tr>
<tr>
<td>Vectors</td>
<td>array</td>
<td>Data type of the input vector</td>
</tr>
<tr>
<td>Matrices</td>
<td>array</td>
<td>Data type of the square input matrix</td>
</tr>
</tbody>
</table>

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation Vector_Elements * Matrix_Elements := Vector_Elements -</td>
</tr>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation Vector_Elements * Vector_Elements := Scalars</td>
</tr>
</tbody>
</table>

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_W );

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,
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Matrix_Multiply_</td>
<td>generic</td>
<td>GVMA</td>
<td>Used to multiply the input vector by the square matrix</td>
</tr>
<tr>
<td>Restricted</td>
<td>function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dot_Product_Multiply_</td>
<td>generic</td>
<td>GVMA</td>
<td>Used to multiply the product of the first operation by the transpose of the input vector</td>
</tr>
<tr>
<td>Restricted</td>
<td>function</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 DESIGN

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:
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS_Opns</td>
<td>package</td>
<td>Package defining Vector Scalar operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only multiply operator is used. Instantiation of GVMA.Vector_Scalar_Operations_Constrained</td>
</tr>
</tbody>
</table>

Subprograms:

The following table summarizes the subroutines which are local to this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiply_VS</td>
<td>function</td>
<td>Function defining the operation Vector Elements * Scalars := Vector Elements. Used in instantiation. Used in instantiation of Vector_Scalar_Operations_Constrained</td>
</tr>
<tr>
<td>Divide_VS</td>
<td>function</td>
<td>Function provided defining operation Vector Elements / Scalars := Scalars. Provided for instantiation. Not used in any computations. Used in instantiation of Vector_Scalar_Operations_Constrained</td>
</tr>
<tr>
<td>VV_Transpose_Multiply</td>
<td>function</td>
<td>Function defining a matrix transpose multiply A Matrices * transpose A Matrices := C Matrices. Instantiation of GVMA.Matrix_Matrix_Transpose_Multiply</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Elements</td>
<td>floating point type</td>
<td>Type of element in the input vector.</td>
</tr>
<tr>
<td>Matrix_Elements</td>
<td>floating point type</td>
<td>Type of element in the square output matrix.</td>
</tr>
<tr>
<td>Scalars</td>
<td>floating point type</td>
<td>Type of element in the output scalar</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension the input vector and the square output matrix</td>
</tr>
<tr>
<td>Vectors</td>
<td>array</td>
<td>Data type of the input vector</td>
</tr>
<tr>
<td>Matrices</td>
<td>array</td>
<td>Data type of the square output matrix</td>
</tr>
</tbody>
</table>

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation Vector_Elements * Scalars := Vector_Elements</td>
</tr>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation Vector_Elements * Vector_Elements := Matrix_Elements</td>
</tr>
</tbody>
</table>

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_Ops is new Vector_Scalar_Operations_Constrained  
    ( Elements1 => Vector_Elements,  
    Element 2 => Vector_Elements,  
    Scalars => Scalars,  
    Indices => Indices,  
    Vectors1 => Vectors,  
    Vectors2 => Vectors  
        "*" => Multiply_VS,  
        "/"  => Divide_VS );

use VS_Ops;

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector_Scalar_Operations_Constrained</td>
<td>generic package</td>
<td>GVMA</td>
<td>Package allowing operation Vectors * Scalars := Vectors.</td>
</tr>
</tbody>
</table>

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix_Matrix_Transpose_Multiply</td>
<td>generic function</td>
<td>GVMA</td>
<td>Function allowing the operation vector * transpose vector := Matrices.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>floating</td>
<td>Type of element in the column matrix's column vector</td>
</tr>
<tr>
<td>Elements</td>
<td>point type</td>
<td></td>
</tr>
<tr>
<td>Indices</td>
<td>discrete</td>
<td>Used to dimension the column matrix and the vector in the</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>column matrix</td>
</tr>
<tr>
<td>Vectors</td>
<td>array</td>
<td>Data type of the vector in the column matrix</td>
</tr>
</tbody>
</table>
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-Ø)

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>in</td>
<td>Vectors</td>
<td>The vector to be set on the specified column</td>
</tr>
<tr>
<td>Active_Column</td>
<td>in</td>
<td>Indices</td>
<td>Value designating which column is to be set</td>
</tr>
</tbody>
</table>

3.3.6.2.9.26.10.1.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Column_Matrices</td>
<td>The resultant column matrix</td>
</tr>
</tbody>
</table>

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:

```pascal
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 \((m \times m)\) and a square \((m \times m)\) matrix, yielding a square matrix \((m \times m)\). The matrix multiplies \((A*B)\) and \((AB^* \text{ transpose } A)\) 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B Matrix</td>
<td>floating point</td>
<td>Type of element in the square input matrix</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td></td>
</tr>
<tr>
<td>C Matrix</td>
<td>floating point</td>
<td>Type of element in the output vector</td>
</tr>
<tr>
<td></td>
<td>point type</td>
<td></td>
</tr>
<tr>
<td>B_Matrices</td>
<td>array</td>
<td>Data type of the square input matrix</td>
</tr>
<tr>
<td>C_Matrices</td>
<td>array</td>
<td>Data type of the output matrix</td>
</tr>
</tbody>
</table>

Subprograms:

The following table describes the generic formal subroutines required by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation Vector_Eleinents *</td>
</tr>
<tr>
<td></td>
<td>B_Matrix_Elements := B_Matrix_Elements</td>
<td></td>
</tr>
</tbody>
</table>

FORMAL PARAMETERS:

The following table describes the formal parameters for this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Column Matrices</td>
<td>The input column matrix</td>
</tr>
<tr>
<td>B</td>
<td>B_Matrices</td>
<td>The input square matrix</td>
</tr>
</tbody>
</table>

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:

```vcl
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_Eleinents(
            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_SYM_T T RANSPOSE 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

#### LOCAL DATA

None.

#### PROCESS CONTROL

Not applicable.

#### PROCESSING

The following describes the processing performed by this part:

```plaintext
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 )  
                    );
                Answer( Col, Row ) := Answer( Row, Col );
            end loop P_Loop;
        end loop M_Loop;
    end if;
    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 \( m \times n \) matrix by an \( n \times p \) matrix, returning an \( m \times p \) matrix.

The result of this operation is defined as follows:

\[
a(i,j) := b(i,k) \times c(k,j)
\]

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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating point type</td>
<td>Data type of elements in left input matrix</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating point type</td>
<td>Data type of elements in right input matrix</td>
</tr>
<tr>
<td>Output_Elements</td>
<td>floating point type</td>
<td>Data type of elements in output matrix</td>
</tr>
<tr>
<td>M_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of left input matrix and output matrix</td>
</tr>
<tr>
<td>N_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of left input matrix and first dimension of right input matrix</td>
</tr>
<tr>
<td>P_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of right input matrix and output matrix</td>
</tr>
<tr>
<td>Left_Matrices</td>
<td>array</td>
<td>Data type of left input matrix</td>
</tr>
<tr>
<td>Right_Matrices</td>
<td>array</td>
<td>Data type of right input matrix</td>
</tr>
<tr>
<td>Output_Matrices</td>
<td>array</td>
<td>Data type of output matrix</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*</code></td>
<td>function</td>
<td>Function defining the operation (_{\text{Left Elements} \ast \text{Right Elements} := \text{Output Elements}})</td>
</tr>
<tr>
<td><code>+</code></td>
<td>function</td>
<td>Function defining the operation (_{\text{Output Elements} + \text{Output Elements} := \text{Output Elements}})</td>
</tr>
</tbody>
</table>

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left_Matrices</td>
<td>In</td>
<td>(m \times n) matrix</td>
</tr>
<tr>
<td>Right</td>
<td>Right_Matrices</td>
<td>In</td>
<td>(n \times p) matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Output_Matrices</td>
<td>N/A</td>
<td>Result matrix</td>
</tr>
</tbody>
</table>

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:

\[ a(i) := b(i,j) \times c(j) \]

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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix_Elements</td>
<td>floating point</td>
<td>Type of elements in the input matrix</td>
</tr>
<tr>
<td>Input_Vector_Elements</td>
<td>floating point</td>
<td>Type of elements in the input vector</td>
</tr>
<tr>
<td>Output_Vector_Elements</td>
<td>floating point</td>
<td>Type of elements in the output vector</td>
</tr>
<tr>
<td>Indices1</td>
<td>discrete type</td>
<td>Used to dimension first dimension of input matrix and to dimension of output vector</td>
</tr>
<tr>
<td>Indices2</td>
<td>discrete type</td>
<td>Used to dimension second dimension of input matrix and to dimension of input vector</td>
</tr>
<tr>
<td>Input_Matrices</td>
<td>array</td>
<td>Data type of input matrix</td>
</tr>
<tr>
<td>Input_Vectors</td>
<td>array</td>
<td>Data type of input vector</td>
</tr>
<tr>
<td>Output_Vectors</td>
<td>array</td>
<td>Data type of output vector</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Matrix_Elements * Input_Vector_Elements := Output_Vector_Elements</td>
</tr>
<tr>
<td>&quot;+&quot;</td>
<td>function</td>
<td>Function defining the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output_Vector_Elements +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output_Vector_Elements :=</td>
</tr>
</tbody>
</table>

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Input_Matrices</td>
<td>In</td>
<td>Matrix to be used as the multiplicand</td>
</tr>
<tr>
<td>Vector</td>
<td>Input_Vectors</td>
<td>In</td>
<td>Vector to be used as the multiplier</td>
</tr>
</tbody>
</table>

3.3.6.2.10.2.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Output_Vectors</td>
<td>Result of performing the matrix-vector multiplication</td>
</tr>
</tbody>
</table>

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:

\[ a(i,j) := b(i) \times c(j) \]

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot; function</td>
<td>Operator defining the multiplication operation</td>
<td></td>
</tr>
<tr>
<td>Left_Vector_Elements * Right_Vector_Elements := Matrix_Elements</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### FORMAL PARAMETERS:

The following table describes this part’s formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left_Vectors</td>
<td>In</td>
<td>m x 1 vector</td>
</tr>
<tr>
<td>Right</td>
<td>Right_Vectors</td>
<td>In</td>
<td>1 x n vector</td>
</tr>
</tbody>
</table>

### 3.3.6.2.10.3.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Matrices</td>
<td>N/A</td>
<td>Result matrix</td>
</tr>
</tbody>
</table>
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`

```pascal
(Left : Left Vectors;
    Right : Right Vectors) return Matrices is

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

Answer : Matrices;

---begin function Vector_Vector_Transpose_Multiply_Restricted
---end 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 \( m \times n \) matrix by the transpose of a \( p \times n \) matrix, returning the resultant \( m \times p \) matrix.

The results of this operation are defined as follows:

\[
a_{(i,j)} := b(i,k) \times c(j,k)
\]

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating point type</td>
<td>Type of elements in left input matrix</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating point type</td>
<td>Type of elements in right input matrix</td>
</tr>
<tr>
<td>Output_Elements</td>
<td>floating point type</td>
<td>Type of elements in output matrix</td>
</tr>
<tr>
<td>M_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of left input matrix and output matrix</td>
</tr>
<tr>
<td>N_Indices</td>
<td>discrete type</td>
<td>Used to dimension second dimension of left and right input matrix</td>
</tr>
<tr>
<td>P_Indices</td>
<td>discrete type</td>
<td>Used to dimension first dimension of right input matrix and second dimension of output matrix</td>
</tr>
<tr>
<td>Left_Matrices</td>
<td>array</td>
<td>Data type of left input matrix</td>
</tr>
<tr>
<td>Right_Matrices</td>
<td>array</td>
<td>Data type of right input matrix</td>
</tr>
<tr>
<td>Output_Matrices</td>
<td>array</td>
<td>Data type of output matrix</td>
</tr>
</tbody>
</table>
Subprograms:
The following table describes the generic formal subroutines required by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Operator used to define the operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Left_Elements * Right_Elements := Output_Elements</td>
</tr>
</tbody>
</table>

FORMAL PARAMETERS:
The following table describes this part's formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left_Matrices</td>
<td>In</td>
<td>Matrix to be used as the multiplicand</td>
</tr>
<tr>
<td>Right</td>
<td>Right_Matrices</td>
<td>In</td>
<td>Matrix whose transpose is to be used as the multiplier</td>
</tr>
</tbody>
</table>

3.3.6.2.10.4.4 LOCAL DATA
Data objects:
The following table describes the data objects maintained by this part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Output_Matrices</td>
<td>N/A</td>
<td>Result matrix being calculated</td>
</tr>
</tbody>
</table>

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:

```pl
separate (General_Vector_Matrix_Algebra)
function Matrix_Matrix_Transpose_Multiply_Restricted
  (Left : Left_Matrices;
   Right : Right_Matrices) return Output_Matrices is
  Answer : Output_Matrices;
```

```pl
end
```
--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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left_Elements</td>
<td>floating point type</td>
<td>Type of elements in left input vector</td>
</tr>
<tr>
<td>Right_Elements</td>
<td>floating point type</td>
<td>Type of elements in right input vector</td>
</tr>
<tr>
<td>Result_Elements</td>
<td>floating point type</td>
<td>Data type of result of dot product</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension input vectors</td>
</tr>
<tr>
<td>Left_Vectors</td>
<td>array</td>
<td>Data type of left input vector</td>
</tr>
<tr>
<td>Right_Vectors</td>
<td>array</td>
<td>Data type of right input vector</td>
</tr>
</tbody>
</table>

Subprograms:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Multiplication operator defining the operation: Left_Elements * Right_Elements := Result_Elements</td>
</tr>
</tbody>
</table>

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left_Vectors</td>
<td>In</td>
<td>First vector in a dot product operation</td>
</tr>
<tr>
<td>Right</td>
<td>Right_Vectors</td>
<td>In</td>
<td>Second vector in a dot product operation</td>
</tr>
</tbody>
</table>

3.3.6.2.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Result_Elements</td>
<td>N/A</td>
<td>Result of dot product operation</td>
</tr>
</tbody>
</table>

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:

```pascal
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

GENERAL 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>floating point type</td>
<td>Type of elements in input and output arrays</td>
</tr>
<tr>
<td>Diagonal_Range</td>
<td>integer type</td>
<td>Used to dimension Diagonal_Matrices</td>
</tr>
<tr>
<td>Indices</td>
<td>discrete type</td>
<td>Used to dimension input and output matrices</td>
</tr>
<tr>
<td>Diagonal_Matrices</td>
<td>array</td>
<td>Data type of diagonal input matrix</td>
</tr>
<tr>
<td>Full_Matrices</td>
<td>array</td>
<td>Data type of full input and output matrices</td>
</tr>
</tbody>
</table>

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_Matrix</td>
<td>Diagonal_Matrices</td>
<td>In</td>
<td>Input diagonal matrix</td>
</tr>
<tr>
<td>F_Matrix</td>
<td>Full_Input_Matrices</td>
<td>In</td>
<td>Input full matrix to be added to the diagonal matrix</td>
</tr>
</tbody>
</table>
3.3.6.2.10.6.4 LOCAL DATA

Data objects:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Full_output_matrices</td>
<td>N/A</td>
<td>Resultant matrix</td>
</tr>
<tr>
<td>Diag_Index</td>
<td>Diagonal_range</td>
<td>N/A</td>
<td>Index into diagonal matrix</td>
</tr>
<tr>
<td>Index</td>
<td>Indices</td>
<td>N/A</td>
<td>Index into full matrix</td>
</tr>
</tbody>
</table>

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:

```plaintext
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;
```
END OF CODE:

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:

\[ c(j) := a(i) \times b(i,j) \]

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:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix_Elements</td>
<td>floating point type</td>
<td>Type of elements in the input matrix</td>
</tr>
<tr>
<td>Input_Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements in the input vector</td>
</tr>
<tr>
<td>Output_Vector_Elements</td>
<td>floating point type</td>
<td>Type of elements in the output vector</td>
</tr>
<tr>
<td>Indices1</td>
<td>discrete type</td>
<td>Used to dimension first dimension of input matrix and to dimension the output vector</td>
</tr>
<tr>
<td>Indices2</td>
<td>discrete type</td>
<td>Used to dimension second dimension of input matrix and to dimension the input vector</td>
</tr>
<tr>
<td>Input_Matrices</td>
<td>array</td>
<td>Data type of input matrix</td>
</tr>
<tr>
<td>Input_Vectors</td>
<td>array</td>
<td>Data type of input vector</td>
</tr>
<tr>
<td>Output_Vectors</td>
<td>array</td>
<td>Data type of output vector</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;*&quot;</td>
<td>function</td>
<td>Function defining the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Input_Vector_Elements * Matrix_Elements :=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output_Vector_Elements</td>
</tr>
<tr>
<td>&quot;+&quot;</td>
<td>function</td>
<td>Function defining the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output_Vector_Elements +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output_Vector_Elements :=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output_Vector_Elements</td>
</tr>
</tbody>
</table>

FORMAL PARAMETERS:

The following table describes this part’s formal parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Input_Vectors</td>
<td>In</td>
<td>M-element vector</td>
</tr>
<tr>
<td>Matrix</td>
<td>Input_Matrices</td>
<td>In</td>
<td>M x N input matrix</td>
</tr>
</tbody>
</table>

3.3.6.2.10.7.4 LOCAL DATA

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>Output Vectors</td>
<td>N/A</td>
<td>Vector being calculated and returned</td>
</tr>
</tbody>
</table>

### 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;
package body Vector_Operations_Unconstrained is

pragma PAGE;

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

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

begin
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 VectorElements is

-- declaration section-
-- declaration section-

Temp      : VectorElements_Squared;

-- begin function Vector_Length
-- 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

raise Dimension_Error;

end function " + ";
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);
R_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

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

end if;

return Answer;

end "+";

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

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

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;
         end loop;
   end loop Row_Loop;

   return Answer;
end "+";

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

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

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;
         end loop;
   end loop Row_Loop;

   return Answer;
end ";";
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;
package body Dynamically_Sparse_Matrix_Operations_Unconstrained is

pragma PAGE;
procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

-- declaration section
-- 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;
-- --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:
    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;
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
    LCol := Left'FIRST(2);
    RCol := 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);

  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;
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(Swap_Row(COL)) + 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
every next row
Add_Identity_Loop:
for INDEX in Row_Indices'SUC(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

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'SUC(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

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--
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));
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;
Rev     := 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;
Left'range(1), Left'range(2)); return Matrices is

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

  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
   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;
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
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
begin
  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

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

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
end Init_Block;
end Diagonal_Matrix_Operations;
package body Vector_Scalar_Operations_Unconstrained is

pragma PAGE;

function "*" (Vector : Vectors2;
               Multiplier : Scalars) return Vectors1 is

   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

   begin function Vector_Scalar_Divide

   begin

      A_Index := Indices1'FIRST;
      V_Index := Indices2'FIRST;
      Process:
      loop
         Answer(A_Index) := Vector(V_Index) / Divisor;
         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 "/";

end Vector_Scalar_Operations_Unconstrained;
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;
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 section-

   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;
               end loop;

               exit Col_Loop when M_Col = Matrix'LAST(2);
               A_Col := Col_Indices2'SUCCE(A_Col);
               M_Col := Col_Indices1'SUCCE(M_Col);

            end loop;

            exit Row_Loop when M_Row = Matrix'LAST(1);
            A_Row := Row_Indices2'SUCCE(A_Row);
            M_Row := Row_Indices1'SUCCE(M_Row);

         end 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_Matrices2;
    Multiplier : Scalars) return Diagonal_Matrices2 is

  -- declaration section-
  -- ----------------------
  --
  Answer : Diagonal_Matrices2;
  Indexl : Diagonal_Rangel;
  Index2 : Diagonal_Range2;

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

  begin
    Indexl := Diagonal_Rangel'FIRST;
    Index2 := Diagonal_Range2'FIRST;
    Process:
      loop
        Answer(Index2) := Matrix(Indexl) * Multiplier;
        exit Process when Indexl = Diagonal_Rangel'LAST;
        Indexl := Diagonal_Rangel'SUCC(Indexl);
        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;
  Indexl : Diagonal_Rangel;
  Index2 : Diagonal_Range2;

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

  begin
    Index1 := Diagonal_Rangel'FIRST;
Index2 := Diagonal_Range2'FIRST;
Process:
  loop
    Answer(Index1) := Matrix(Index2) / Divisor;
    exit Process when Index1 = Diagonal_Rangel'LAST;
    Index1 := Diagonal_Rangel'SUCC(Index1);
    Index2 := Diagonal_Range2'SUCC(Index2);
  end loop Process;

return Answer;
end "/";
pragma PAGE;

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

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

N_Left := Left_Col_Indices'FIRST;
N_Right := Right_Row_Indices'FIRST;
N_Loop:
loop

Answer(M_Answer, P_Answer) := 0.0;
exit N_Loop when N_Left = Left_Col_Indices'LAST;
N_Left := Left_Col_Indices'SUC(N_Loop);
N_Right := Right_Row_Indices'SUC(N_Right);
exit P_Loop when P_Right = Right_Col_Indices'LAST;
P_Right := Right_Col_Indices'SUC(P_Loop);
P_Answer := Output_Col_Indices'SUC(P_Answer);
end loop P_Loop;

Answer(M_Answer, P_Answer) :=
Answer(M_Answer, P_Answer) +
Left(M_Left, N_Left) * Right(N_Right, P_Right);

end loop M_Loop;

end function "*";
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: \([m \times n] \times [n \times p] = [m \times p]\)
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'SUCCESE(N_Matrix);
              N_Vector := Input_Vector_Indices'SUCCESE(N_Vector);
            end loop N_Loop;
            exit M_Loop when M_Matrix = Row_Indices'LAST;
            M_Matrix := Row_Indices'SUCCESE(M_Matrix);
            M_Answer := Output_Vector_Indices'SUCCESE(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 be requested: \([m \times n] \times [n \times 1] = [m \times 1]\)
if not (Input_Matrices'LENGTH(2) = Input_Vectors'LENGTH and
     Input_Matrices'LENGTH(1) = Output_Vectors'LENGTH) then -- "n's"
    -- dimensions are incompatible
    raise Dimension_Error;
end if;
end Matrix_Vector_Multiply_Unrestricted;
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 \[ m \times 1 \times 1 \times n \] = \[ m \times n \]
if not (Left_Vectors'LENGTH = Matrices'LENGTH(1) and Right_Vectors'LENGTH = Matrices'LENGTH(2)) then
   raise Dimension_Error;
end if;
end Vector_Vector_Transpose_Multiply_Unrestricted;
package body Matrix_MatrTx_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_\text{Left} : Left_Row_Indices;
    N_\text{Left} : Left_Col_Indices;
    N_\text{Right} : Right_Col_Indices;
    P_Answer : Output_Col_Indices;
    P_\text{Right} : Right_Row_Indices;

    begin
        M_Answer := Output_Row_Indices'FIRST;
        M_\text{Left} := Left_Row_Indices'FIRST;
        \text{M\_Loop}:
        -- loop
            P_Answer := Output_Col_Indices'FIRST;
            P_\text{Right} := Right_Row_Indices'FIRST;
            \text{P\_Loop}:
            -- loop
                Answer(M_Answer, P_Answer) := 0.0;
                N_\text{Left} := Left_Col_Indices'FIRST;
                N_\text{Right} := Right_Col_Indices'FIRST;
                \text{N\_Loop}:
                -- loop
                    Answer(M_Answer, P_Answer) :=
                        Answer(M_Answer, P_Answer) +
                        Left(M_\text{Left}, N_\text{Left}) * Right(P_\text{Right}, N_\text{Right});
                    exit \text{N\_Loop when N_\text{Left} = Left_Col_Indices'LAST};
                    N_\text{Left} := Left_Col_Indices'SUC\(C(N_\text{Left});
                    N_\text{Right} := Right_Col_Indices'SUC\(C(N_\text{Right});
                \end \text{N\_Loop};
            exit \text{P\_Loop when P_\text{Answer} = Output_Col_Indices'LAST};
            P_Answer := Output_Col_Indices'SUC\(C(P_\text{Answer});
            P_\text{Right} := Right_Row_Indices'SUC\(C(P_\text{Right});
        \end \text{P\_Loop};
    \end \text{M\_Loop};
exit M_Loop when M_Answer = Output_Row_Indices'LAST;
M_Answer := Output_Row_Indices'SUC(P)(M_Answer);
M_Left  := Left_Row_Indices'SUC(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;
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;

Assign_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;

  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
---------------------------------
begine

-- --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;
This report has been delimited and cleared for public release under DoD Directive 5200.20 and no restrictions are imposed upon its use and disclosure.

Distribution Statement A

Approved for public release; distribution unlimited.
package body Vector_Operations_Constrained is

pragma PAGE;

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

    -- declaration section-
    -- declaration section-
    Answer : Vectors;

    -- begin function "+
    -- 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-
    -- declaration section-
    Answer : Vectors;

    -- begin function "-
    -- 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
  begin -- declaration section-
  begin
    Temp : Vector_Elements_Squared;
    begin
      begin
        Temp := 0.0;
        for INDEX in Indices loop
          Temp := Temp +
            Input(INDEX) * Input(INDEX);
        end loop;
        return Sqrt(Temp);
      end;
    end;
  end Vector_Length;
  end;

pragma PAGE;
function Dot_Product (Left : Vectors;
                      Right : Vectors) return Vector_Elements_Squared is
  begin -- declaration section
  begin
    Answer : Vector_Elements_Squared;
    begin
      begin
        Answer := 0.0;
        for INDEX in Indices loop
          Answer := Answer + Left(INDEX) * Right(INDEX);
        end loop;
        return Answer;
      end;
    end;
  end Dot_Product;
end Vector_Operations_Constrained;
package body Matrix_Operations_Constrained is

pragma PAGE;

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

  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

  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;
package body Dynamically_Sparse_Matrix_Operations_Constrained is

pragma PAGE;
procedure Set_To_Identity_Matrix (Matrix : out Matrices) is

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;
CAMP Software Detailed Design Document

begin

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

Answer : Matrices;
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

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 "-";
Answer(Row, COL) := 0.0;
else
  Answer(Row, COL) := - Right(Row, COL);
end if;
elif 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

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
SCol := 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);

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

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
  end loop

return Answer;
end Subtract_From_Identity;

pragma PAGE;

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

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;
    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
  end loop

return Answer;
end Subtract_From_Identity;

pragma PAGE;

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

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
  end loop

return Answer;
end Subtract_From_Identity;

pragma PAGE;
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

begin

Answer : Matrices;
Row : Row_Indices;
S_Col : Col_Indices;
S_Row : Row_Indices;

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_Elelnets;

-- 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;
package body Vector_Scalar_Operations_Constrained is

pragma PAGE;

function "*" (Vector : Vectors2;
Multiplier : Scalars) return Vectors1 is

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

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;
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 "/";
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

  Answer : Output_Matrices;

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

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 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 MatrixTranspose_Multiply_Restricted
(Left : Left Matrices;
Right : Right Matrices) return Output Matrices is

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

Answer : Output Matrices;

-- begin function Matrix MatrixTranspose_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 MatrixTranspose_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
  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;
package body Vector_MatrTx_MultTply_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
-- make sure package has been instantiated with the correct dimensions;
-- the following dimensions are expected: [1x m] * [m x n] = [1 x n]

if Input_Vectors'LENGTH /= Input_Matrices'LENGTH(1) then
   raise Dimension_Error;
end if;

end Vector_Matrix_Multiply_Unrestricted;
separate (General_Vector_Matrix_Algebra)
function Vector_Matric_Multiply_Restricted
    (Vector : Input_Vectors;
     Matrix : Input_Matrices) return Output_Vectors is

    Answer : Output_Vectors := (others => 0.0);

    begin function Vector_Matric_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 function Vector_Matric_Multiply_Restricted;
function Sparse_Left_Multiply(Left : A_Elements;
Right : B_Elements ) return A_Elements is
  Answer : A_Elements;
begi
  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;
begi
  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_Multiply_Restricted
  ( Left_Elements => A_Elements,
    Right_Elements => B_Elements,
    Output_Elements => A_Elements,
    M_Indices => M_Indices,
    N_Indices => N_Indices,
    P_Indices => P_Indices,
    Left_Matrices => A_Matrices,
    Right_Matrices => B_Matrices,
    Output_Matrices => A_Matrices,
    "*" => Sparse_Left_Multiply );

function Matrix_Transpose_Multiply is new 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 => P_Indices,
    Left_Matrices => A_Matrices,
    Right_Matrices => A_Matrices,
    Output_Matrices => C_Matrices,
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,
"*" => 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_Vecto: _Multiply( Left => Partial_Answer, 
Right => A );

return Answer;

dend Aba_Transpose;

dend Aba_Trans_Vector_Sq_Matrix;
package body Aba_Trans_Vector_Scalar is

-- -- Operators provided for instantiations -

function Multiply_Vs( Left    : Vector_Elemenents;
                      Right  : Scalars ) return Vector_Elemenents is
begin
  return Left * Vector_Elemenents( 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_Elemenents;
                    Right : Scalars ) return Vector_Elemenents is
begin
  return Left;
end Divide_Vs;

function Multiply_Vv( Left : Vector_Elemenents;
                      Right : Vector_Elemenents ) return Matrix_Elemenents is
begin
  return Matrix_Elemenents( Left ) * Matrix_Elemenents( Right );
end Multiply_Vv;

-- -- Instantiations for ABA transpose -

package Vs_Opns is new Vector_Scalar_Operations_Constrained
  ( Elements1 => Vector_Elemenents,
    Elements2 => Vector_Elemenents,
    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_Elemenents    => Vector_Elemenents,
    Right_Vector_Elemenents  => Vector_Elemenents,
    Matrix_Elemenents        => Matrix_Elemenents,
    Indices1                 => Indices,
    Indices2                 => Indices,
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;
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;
end Aba_Transpose;
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_Elemenets(
        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_Elemenets(
        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
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).

<table>
<thead>
<tr>
<th>Technical Report Number</th>
<th>AD Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 88-18-Vol-4</td>
<td>ADB 120 251</td>
</tr>
<tr>
<td>2. 88-18-Vol-5</td>
<td>ADB 120 252</td>
</tr>
<tr>
<td>3. 88-18-Vol-6</td>
<td>ADB 120 253</td>
</tr>
<tr>
<td>4. 88-25-Vol-1</td>
<td>ADB 120 309</td>
</tr>
<tr>
<td>5. 88-25-Vol-2</td>
<td>ADB 120 310</td>
</tr>
<tr>
<td>6. 88-62-Vol-1</td>
<td>ADB 129 568</td>
</tr>
<tr>
<td>7. 88-62-Vol-2</td>
<td>ADB 129 569</td>
</tr>
<tr>
<td>8. 88-62-Vol-3</td>
<td>ADB 129 570</td>
</tr>
<tr>
<td>9. 85-93-Vol-1</td>
<td>ADB 102-654</td>
</tr>
<tr>
<td>10. 85-93-Vol-2</td>
<td>ADB 102-655</td>
</tr>
<tr>
<td>11. 85-93-Vol-3</td>
<td>ADB 102-656</td>
</tr>
<tr>
<td>12. 88-18-Vol-1</td>
<td>ADB 120 248</td>
</tr>
<tr>
<td>13. 88-18-Vol-2</td>
<td>ADB 120 249</td>
</tr>
<tr>
<td>14. 88-18-Vol-7</td>
<td>ADB 120 254</td>
</tr>
<tr>
<td>15. 88-18-Vol-8</td>
<td>ADB 120 255</td>
</tr>
<tr>
<td>16. 88-18-Vol-9</td>
<td>ADB 120 256</td>
</tr>
<tr>
<td>17. 88-18-Vol-10</td>
<td>ADB 120 257</td>
</tr>
<tr>
<td>18. 88-18-Vol-11</td>
<td>ADB 120 258</td>
</tr>
<tr>
<td>19. 88-18-Vol-12</td>
<td>ADB 120 259</td>
</tr>
</tbody>
</table>

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
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