

```
-- *****  
-- *  
-- * Matrix_Package  
-- *  
-- *****  
generic  
  type ELEMENT is digits <>;  
  type INDEX   is range <>;  
  type VECTOR is array (INDEX range <>) of ELEMENT ;  
  type MATRIX is array (INDEX range <>, INDEX range <>) of ELEMENT  
package MATRIX_PACKAGE is  
  -- | Purpose  
  -- | This package is a general purpose matrix package. It  
  -- | types VECTOR and MATRIX, and contains functions to pe  
  -- | matrix algebra operations.  
  -- |  
  -- | Initialization Exceptions (none)  
  -- |  
  -- | Notes  
  -- |   Not all MIL-HDBK-      PDL annotations are used in t  
  -- | due to its simplicity.  
  -- |  
  -- | Modifications  
  -- | Author: Dr. Roger Lee, Naval Air Development Center  
  -- |         Art Adamson, Consultant  
  -- |         Louis Granger, Generic part  
  -- Types  
  
  subtype VEC T is VECTOR (INDEX range .. ) ;  
  subtype VEC T is VECTOR (INDEX range .. ) ;  
  
  type MATR T is array (INDEX range <>) of VEC T;  
  
  -- Exceptions  
  INCOMPARABLE_DIMENSION : exception; -- the dimension of matr  
  -- or vectors to be operated are
```

Untitled

```
-- incomparable
SINGULAR : exception;      -- matrix to be inverted is singular

-- Operations
function TRANSPOSE (A : MATRIX) return MATRIX ; -- transpose of
function TRANSPOSE (A : VECTOR) return VECTOR ; -- transpose of
function "+" (A : VECTOR; B : VECTOR) return VECTOR ; -- sum of
function "+" (A : MATRIX; B : MATRIX) return MATRIX ; -- sum of
function "+" (A : ELEMENT; B : VECTOR) return VECTOR ;
-- float added to, each term of matrix
function "+" (A : VECTOR; B : MATRIX) return MATRIX ;
-- Vec T added to, each term of MATR T
function "+" (A : MATRIX; B : MATRIX) return MATRIX ;
-- Corressponding terms added.
function "-" (A : VECTOR; B : VECTOR) return VECTOR ;
-- difference of vector
function "-" (A : MATRIX; B : MATRIX) return MATRIX ;
-- difference of matrix
function "*" (A : ELEMENT; B : VECTOR) return VECTOR ;
-- scalar, vector multiplication
function "*" (A : VECTOR; B : ELEMENT) return VECTOR ;
-- vector, scalar multiplication
function "*" (A : VECTOR; B : VECTOR) return ELEMENT ;
-- inner(dot) product of two vectors
function "*" (A : MATRIX; B : VECTOR) return VECTOR ;
-- matrix,column vector multiplication
function MAT MULT (UL : MATRIX; UR : MATRIX; BL : MATRIX; BR :
      B : VECTOR) return VECTOR ;
-- large matrix broken into smaller ones, column vector
-- (upper left, upper right, bottom left, bottom right--a
function "*" (A : VECTOR; B : MATRIX) return VECTOR ;
-- row vector,matrix multiplication
function "*" (A : ELEMENT; B : MATRIX) return MATRIX ;
-- scalar, matrix multiplication
function "*" (A : MATRIX; B : ELEMENT) return MATRIX ;
```

Untitled

```
-- matrix, scalar multiplication
function "*" (A : MATRIX; B : MATRIX) return MATRIX ;
-- matrix, matrix multiplication
function "*" (A : ELEMENT; B : MATR T) return MATR T ;
-- Multiplies each term of a MATR T by a float
function "*" (A : VEC T; B : MATR T) return VECTOR ;
-- Dot product of each term of MATR T by a VEC T, return ;
function "*" (A : VECTOR; B : MATR T) return MATR T ;
--Multiplies each term of VEC T by a corresponding float
function "***" (A : MATRIX; P : INTEGER) return MATRIX;
-- square matrix raised to integer power
-- if P = - , we invert the matrix
function "***" (A : VECTOR; B : VECTOR) return VECTOR ;
-- A X B = ab sin(theta) a direction
--perpendicular to plane of A & B.
function JCROSS (A : VEC T) return VEC T ;
--Rotates Vec T degrees CW.
function JCROSS (A : MATR T) return MATR T ;
--Rotates Vec T's degrees CW.
function ROTX (A : VEC T) return VEC T ;
--Rotates Vec T degrees about the X axis.
function ROTY (A : VEC T) return VEC T ;
--Rotates Vec T degrees about the Y axis.
function AXBDOTJ (A : VEC T; B : VEC T) return ELEMENT;
--Gets magnitude of A cross B for D vectors.
function GETTAN (A : VEC T; B : VEC T) return ELEMENT;
--Gets TAN(THETA) between D vectors.

end MATRIX_PACKAGE;

-- *****
-- *
-- * Matrix_Package
-- *
-- *****
```

Untitled

```
with TEXT_IO;
use TEXT_IO;
package body MATRIX_PACKAGE is
  --| Notes
  --| Modifications by Art Adamson --
  -- Mod by A. P. Adamson..July      ..Added cross product c
  -- The added function Vector ** Vector return Vector is lin
  -- -D vectors. Vector = Vector cross Vector .
  -- Mod by A. P. Adamson..Oct.      ,      ..Added VEC T, a D
  -- Mod by A. P. Adamson..Oct.      ,      ..Added VEC T, a D
  -- Mod by A. P. Adamson..Oct.      ,      ..Added MATR T, a VE
  -- elements of VEC T rather than float.
  -- Mod by A. P. Adamson..Oct.      ,      ..Added +, an operat
  -- to each term of a VECTOR.
  -- Mod by A. P. Adamson..Oct.      ,      ..Added *, an operat
  -- each VEC T of a MATR T by a single float.
  -- Mod by A. P. Adamson..Oct.      ,      ..Added *, an operat
  -- each VEC T of a MATR T by a single VEC T.
  -- Mod by A. P. Adamson..Oct.      ,      ..Added JCROSS, an
  -- product a fictitious unit vector j in y direction with
  -- components only in the x and z directions. Result is a
  -- allow direct translation of certain d vector formulas
  -- Net effect is to rotate the vec t degrees CW.
  -- Mod by A. P. Adamson..Oct.      ,      ..Added + , an oper
  -- VEC T to each term of a MATR T.
  -- Mod by A. P. Adamson..Oct.      ,      ..Added * , an oper
  -- each term of a MATR T by a corresponding float term fr
  -- Mod by A. P. Adamson..Oct.      ,      ..Added + , an oper
  -- corresponding VEC T's of two MATR T and return MATR T.
  -- Mod by A. P. Adamson..Nov.      ,      ..Added ROTX and ROT
  -- rotate a VEC T degrees about the X axis or the Y a
  -- Mod by A. P. Adamson..Nov.      ,      ..Added MAT MULT of
  -- column multiply square input matricees (the corner
  -- ger
  -- matrix) by a column vector and return a column vector.
  -- rix
  -- is too large for the memory capacity.
  -- Mod by A. P. Adamson..Jan.      ,      ..Added aXbDOTj ope
  -- give scalar = mag of A cross B for VEC T vectors.
```

Untitled

```
-- VEC T has no third dimensoin so the cross product is r
-- Mod by A. P. Adamson..Jan. , ..Added GETTAN oper
-- AN
-- of angle theta between VEC T vectors.
-- Mod by A. P. Adamson..Feb. , ..Added GETTAN ope
--for divide by when angle = PI/ .
```

```
function TRANSPOSE (A : MATRIX) return MATRIX is
  B : MATRIX (A'FIRST ( ) .. A'LAST ( ), A'FIRST ( ) .. A'LAST ( ))
  -- *****
  -- This function performs the tranpose of input matrix
  -- *****
begin
  for I in A'RANGE ( ) loop
    for J in A'RANGE ( ) loop
      B (I, J) := A (J, I) ;
    end loop ;
  end loop ;
  return B ;
end TRANSPOSE;
```

```
function TRANSPOSE (A : VECTOR) return VECTOR is
  -- *****
  -- This function returns the transpose of a vector. In
  -- a vector is always stored as one-dimensional array
  -- there is no difference between row vector and column vector
  -- Thus, this function just returns the input vector
  -- *****
begin
  return A;
end TRANSPOSE;
```

```
function "+" (A : VECTOR; B : VECTOR) return VECTOR is
  C : VECTOR (A'FIRST .. A'LAST) ;
```

Untitled

```
-- *****
-- This function performs the addition of vector A
-- resulting in a vector. Comparability of dimensi
-- *****

begin
  if A'FIRST /= B'FIRST or A'LAST /= B'LAST then
    raise INCOMPARABLE_DIMENSION;
  end if ;
  for I in A'RANGE loop
    C (I) := A (I) + B (I) ;
  end loop ;
  return C ;
end "+";
---
```

```
function "+" (A : ELEMENT; B : VECTOR) return VECTOR is
  C : VECTOR (B'FIRST .. B'LAST) ;
  -- *****
  -- This function performs the addition of a FLOAT I
  -- of vector B resulting in a vector.
  -- *****

begin
  for I in B'RANGE loop
    C (I) := A + B (I) ;
  end loop ;
  return C ;
end "+";
---
```

```
function "+" (A : MATRIX; B : MATRIX) return MATRIX is
  C : MATRIX (A'FIRST ( ) .. A'LAST ( ), A'FIRST ( ) .. A'LAST ( )) ;
  -- *****
  -- This function performs the addition of matrix A
  -- resulting in a matrix. Comparability of dimensi
  -- *****

begin
```

Untitled

```
if (A'FIRST ( ) /= B'FIRST ( ) or A'LAST ( ) /= B'LAST ( )
    or (A'FIRST ( ) /= B'FIRST ( ) or A'LAST ( ) /= B'LAST
    raise INCOMPARABLE_DIMENSION;
end if ;
for I in A'RANGE ( ) loop
    for J in A'RANGE ( ) loop
        C (I, J) := A (I, J) + B (I, J) ;
    end loop ;
end loop ;
return C ;
end "+";
--
function "+" (A : VEC T; B : MATR T) return MATR T is
    TEMP : MATR T (B'RANGE);
    -- *****
    -- Vec T added to, each term of MATR T
    -- *****
begin
    for I in B'RANGE loop
        TEMP (I) := A + B (I);
    end loop;
    return TEMP;
end "+";
--
function "+" (A : MATR T; B : MATR T) return MATR T is
    TEMP : MATR T (B'RANGE);
    --*****
    -- Vec T added to, each term of MATR T
    --*****
begin
    for I in B'RANGE loop
        TEMP (I) := A (I) + B (I);
    end loop;
    return TEMP;
```

Untitled

```
end "+";
--
function "-" (A : VECTOR; B : VECTOR) return VECTOR is
  C : VECTOR (A'FIRST .. A'LAST) ;
  -- *****
  -- This function performs the subtraction of vector
  -- resulting in a vector. Comparability of dimension
  -- *****
begin
  if A'FIRST /= B'FIRST or A'LAST /= B'LAST then
    raise INCOMPARABLE_DIMENSION;
  end if ;
  for I in A'RANGE loop
    C (I) := A (I) - B (I) ;
  end loop ;
  return C ;
end "-";
---
function "-" (A : MATRIX; B : MATRIX) return MATRIX is
  C : MATRIX (A'FIRST ( ) .. A'LAST ( ), A'FIRST ( ) .. A'LAST ( )) ;
  -- *****
  -- This function performs the subtraction of matrix
  -- resulting in a matrix. Comparability of dimension
  -- *****
begin
  if (A'FIRST ( ) /= B'FIRST ( ) or A'LAST ( ) /= B'LAST ( )
    or (A'FIRST ( ) /= B'FIRST ( ) or A'LAST ( ) /= B'LAST ( )
    raise INCOMPARABLE_DIMENSION;
  end if ;
  for I in A'RANGE ( ) loop
    for J in A'RANGE ( ) loop
      C (I, J) := A (I, J) - B (I, J) ;
    end loop ;
  end loop ;
end loop ;
```

```

    return C ;
end "-";
---
function "*" (A : ELEMENT; B : VECTOR) return VECTOR is
    C : VECTOR (B'FIRST .. B'LAST);
    -- *****
    -- This function performs the scalar multiplication
    -- number A and a vector B resulting in a vector.
    -- *****
begin
    for I in B'RANGE loop
        C (I) := A * B (I);
    end loop;
    return C ;
end "*";
---
function "*" (A : VECTOR; B : ELEMENT) return VECTOR is
begin
    -- *****
    -- This function performs the scalar multiplication
    -- a floating number B resulting in a vector.
    -- *****
    return B * A;
end "*";
---
function "*" (A : VECTOR; B : VECTOR) return ELEMENT is
    S : ELEMENT := . ;
    -- *****
    -- This function performs the inner (dot) product of
    -- A and B resulting in a floating number.
    -- Comparability of dimensions is checked.
    -- *****
begin
    if A'FIRST /= B'FIRST or A'LAST /= B'LAST then

```

```

        raise INCOMPARABLE_DIMENSION;
    end if ;
    for I in A'RANGE loop
        S := S + A (I) * B (I) ;
    end loop ;
    return S ;
end "*" ;
---
```

```

function "*" (A : MATRIX; B : VECTOR) return VECTOR is
    C : VECTOR (A'FIRST ( ) .. A'LAST ( ));
    SUM : ELEMENT;
    -- *****
    -- This function performs the multiplication of a n
    -- vector B resulting in a column vector.
    -- Comparability of dimensions is checked.
    -- *****
    -- **

begin
    if A'FIRST ( ) /= B'FIRST or A'LAST ( ) /= B'LAST then
        raise INCOMPARABLE_DIMENSION;
    end if ;
    for I in A'RANGE ( ) loop
        SUM := . ;
        for K in A'RANGE ( ) loop
            SUM := SUM + A (I, K) * B (K);
        end loop;
        C (I) := SUM;
    end loop ;
    return C ;
end "*" ;
---
```

```

function MAT MULT (UL : MATRIX; UR : MATRIX; BL : MATRIX; BR :
        return VECTOR is
    C : VECTOR (B'FIRST .. B'LAST);

```

Untitled

```

SUMT, SUMB : ELEMENT;
-- *****
-- This function performs the multiplication of a n
-- smaller ones due to memory limitations, and a c
-- resulting in a column vector. Comparability of c
-- partially checked.
-- *****

begin
  if UL'LENGTH ( ) /= UR'LENGTH ( ) or
     UL'LENGTH ( ) /= BL'LENGTH ( ) or
     UL'LENGTH ( ) /= BR'LENGTH ( ) or
     UL'LENGTH ( ) /= UR'LENGTH ( ) or
     UL'LENGTH ( ) /= BL'LENGTH ( ) or
     UL'LENGTH ( ) /= BR'LENGTH ( ) or
     UL'LENGTH ( ) /= UL'LENGTH ( ) then
    raise INCOMPARABLE_DIMENSION;
  end if;

  if UL'FIRST ( ) /= B'FIRST or UL'LAST ( ) /= B'LAST / then
    raise INCOMPARABLE_DIMENSION;
  end if;

  if UL'FIRST ( ) /= B'FIRST or
     UL'LAST ( ) /= B'LAST / or
     UL'FIRST ( ) /= B'FIRST or
     UL'LAST ( ) /= B'LAST / or
     UL'LAST ( ) /= UL'LAST ( )
  then
    raise INCOMPARABLE_DIMENSION;
  end if ;

  for I in UL'RANGE ( ) loop
    SUMT := . ;
    SUMB := . ;
    for K in UL'RANGE ( ) loop
```

Untitled

```
SUMT := SUMT + UL ( I, K ) * B ( K ) + UR ( I, K )
SUMB := SUMB + BL ( I, K ) * B ( K ) + BR ( I, K )

end loop;
C ( I ) := SUMT;
C ( I + UL'LAST ( ) ) := SUMB;

end loop ;
return C ;
end MAT MULT;
---
```

```
function "*" (A : VECTOR; B : MATRIX) return VECTOR is
  C : VECTOR (B'FIRST ( ) .. B'LAST ( ) );
  SUM : ELEMENT;
  -- *****
  -- This function performs the multiplication of a
  -- matrix B resulting in a row vector.
  -- Comparability of dimensions is checked.
  -- *****
begin
  if A'FIRST /= B'FIRST ( ) or A'LAST /= B'LAST ( ) then
    raise INCOMPARABLE_DIMENSION;
  end if ;
  for J in B'RANGE ( ) loop
    SUM := . ;
    for K in A'RANGE loop
      SUM := SUM + A ( K ) * B ( K, J );
    end loop;
    C ( J ) := SUM;
  end loop ;
  return C ;
end "*" ;
---
```

```
function "*" (A : ELEMENT; B : MATRIX) return MATRIX is
  C : MATRIX (B'FIRST ( ) .. B'LAST ( ), B'FIRST ( ) ..
  -- *****
```

Untitled

```
-- This function performs the scalar multiplication
-- a floating number A resulting in a matrix.
-- *****

begin
  for I in B'RANGE ( ) loop
    for J in B'RANGE ( ) loop
      C (I, J) := A * B (I, J);
    end loop ;
  end loop ;
  return C ;
end "*";
---
```

```
function "*" (A : MATRIX; B : ELEMENT) return MATRIX is
  C : MATRIX (A'FIRST ( ) .. A'LAST ( ), A'FIRST ( ) .. A'LAST ( ))
  -- *****
  -- This function performs the scalar multiplication
  -- by a floating number B resulting in a matrix.
  -- *****

begin
  return B * A ;
end "*";
---
```

```
function "*" (A : MATRIX; B : MATRIX) return MATRIX is
  C : MATRIX (A'FIRST ( ) .. A'LAST ( ), B'FIRST ( ) .. B'LAST ( ))
  SUM : ELEMENT;
  -- *****
  -- This function performs the multiplication of matrices
  -- resulting in a matrix. Comparability of dimensions is
  -- *****

begin
  if A'FIRST ( ) /= B'FIRST ( ) or A'LAST ( ) /= B'LAST ( ) then
    raise INCOMPARABLE_DIMENSION;
  end if ;
  for I in A'RANGE ( ) loop
    for J in B'RANGE ( ) loop
```

Untitled

```
SUM := . ;
for K in A'RANGE ( ) loop
    SUM := SUM + A (I, K) * B (K, J);
end loop;
C (I, J) := SUM;
end loop ;
end loop ;
return C ;
end "*" ;
---
```

```
function "*" (A : ELEMENT; B : MATR T) return MATR T is
    C : MATR T (B'FIRST ( ) .. B'LAST ( ));
    -- *****
    -- This function performs the multiplication of each
    -- MATR T by a FLOAT resulting in a MATR T.
    -- *****
begin
    for I in B'RANGE ( ) loop
        C (I) := A * B (I);
    end loop ;
    return C ;
end "*" ;
---
```

```
function "*" (A : VEC T; B : MATR T) return VECTOR is
    C : VECTOR (B'FIRST ( ) .. B'LAST ( ));
    -- *****
    -- This function performs the DOT PRODUCT of each
    -- MATR T by a VEC T resulting in a VECTOR.
    -- *****
begin
    for I in B'RANGE ( ) loop
        C (I) := A * B (I);
    end loop ;
    return C ;
end "*" ;
```

```

---
function "*" (A : VECTOR; B : MATR T) return MATR T is
  TEMP : MATR T (B'RANGE);
  -- *****
  -- This function multiplies each VEC T of a MATR T
  -- the corresponding term of a VECTOR resultir
  -- *****

begin
  if A'FIRST /= B'FIRST or A'LAST /= B'LAST then
    raise INCOMPARABLE_DIMENSION;
  end if;
  for I in B'RANGE loop
    TEMP (I) := A (I) * B (I);
  end loop;
  return TEMP;
end "*";

function "**" (A : MATRIX; P : INTEGER) return MATRIX is
  B, C : MATRIX (A'FIRST ( ) .. A'LAST ( ), A'FIRST
  I_PIVOT, J_PIVOT : INDEX range A'FIRST ( ) .. A'LAST
  BIG_ENTRY, TEMP, EPSILON : ELEMENT ;
  L, M : array (A'RANGE ( )) of INDEX ;
  -- *****
  -- This function performs the square matrix operati
  -- raise to integer power P ". When P is negative , sa
  --  $A^{*(-N)} = (\text{inverse}(A))^{*N}$  , that is, the inverse
  -- power N . In this case, matrix A must be non-si
  -- Exceptions will be raised if the matrix A is not
  -- or if matrix A is singular.
  -- *****

begin
  if A'FIRST ( ) /= A'FIRST ( ) or A'LAST ( ) /= A'LAST ( ) then
    -- if not a square matrix
    raise INCOMPARABLE_DIMENSION ;

```

```

end if;

if P =  then
  --& B = identity matrix
  for I in A'RANGE ( ) loop
    for J in A'RANGE ( ) loop
      if I /= J then
        B (I, J) :=  . ;
      else
        B (I, J) :=  . ;
      end if;
    end loop;
  end loop;
  return B;
end if ;
B := A ;
if P >  then
  --& B = A multiplied itself for P times
  for I in  .. P -  loop
    B := B * A ;
  end loop ;
  return B ;
end if;
-- P is negative, find inverse first
-- initiate the row and column interchange informatio
for K in B'RANGE ( ) loop
  L (K) := K ; -- row interchange information
  M (K) := K ; -- column interchange information
end loop;
-- major loop for inverse
for K in B'RANGE ( ) loop
  -- & search for row and column index I_PIVOT, J_I
  -- & both in (K .. B'LAST( ) ) for maximum B(I,J)
  -- & in absolute value :BIG_ENTRY

```

Untitled

```
BIG_ENTRY := . ;
--
-- check matrix singularity
--
for I in K .. B'LAST ( ) loop
    for J in K .. B'LAST ( ) loop
        if abs (B (I, J)) > abs (BIG_ENTRY) then
            BIG_ENTRY := B (I, J) ;
            I_PIVOT := I ;
            J_PIVOT := J ;
        end if;
    end loop;
end loop;
if K = B'FIRST ( ) then
    if BIG_ENTRY = . then
        raise SINGULAR;
    else
        EPSILON := ELEMENT (A'LENGTH ( )) * abs (
    end if;
else
    if abs (BIG_ENTRY) < EPSILON then
        raise SINGULAR ;
    end if;
end if;

-- interchange row and column

--& interchange K-th and I_PIVOT-th rows
if I_PIVOT /= K then
    for J in B'RANGE ( ) loop
        TEMP := B (I_PIVOT, J);
        B (I_PIVOT, J) := B (K, J) ;
        B (K, J) := TEMP ;
    end loop;
    L (K) := I_PIVOT ;
```

Untitled

```
end if;
--& interchange K-th and J_PIVOT-th columns
if J_PIVOT /= K then
    for I in B'RANGE ( ) loop
        TEMP := B (I, J_PIVOT) ;
        B (I, J_PIVOT) := B (I, K) ;
        B (I, K) := TEMP ;
    end loop ;
    M (K) := J_PIVOT ;
end if ;

--& divide K-th column by minus pivot (-BIG_ENTRY)

for I in B'RANGE ( ) loop
    if I /= K then
        B (I, K) := B (I, K) / ( - BIG_ENTRY) ;
    end if;
end loop ;

-- reduce matrix row by row

for I in B'RANGE ( ) loop
    if I /= K then
        for J in B'RANGE ( ) loop
            if J /= K then
                B (I, J) := B (I, J) + B (I, K) ;
            end if ;
        end loop ;
    end if ;
end loop ;

--& divide K-th row by pivot

for J in B'RANGE ( ) loop
    if J /= K then
```

Untitled

```

        B (K, J) := B (K, J) / BIG_ENTRY ;
    end if ;
end loop ;
B (K, K) := . / BIG_ENTRY ;
end loop ; -- end of major inverse loop

-- final column and row interchange to obtain
-- inverse of A, i.e. A**(- )

for K in reverse B'RANGE ( ) loop
    -- column interchange
    J_PIVOT := L (K) ;
    if J_PIVOT /= K then
        --& interchange B(I,J_PIVOT) and B(I,K) for ea
        for I in B'RANGE ( ) loop
            TEMP := B (I, J_PIVOT) ;
            B (I, J_PIVOT) := B (I, K) ;
            B (I, K) := TEMP ;
        end loop ;
    end if ;
    -- row interchange
    I_PIVOT := M (K) ;
    if I_PIVOT /= K then
        --& INTECHANGE B(I_PIVOT,J) and B(K,J) for ea
        for J in B'RANGE ( ) loop
            TEMP := B (I_PIVOT, J) ;
            B (I_PIVOT, J) := B (K, J) ;
            B (K, J) := TEMP ;
        end loop ;
    end if ;
end loop ;

-- inverse of A is obtained and stored in B
-- now ready to handle the negative power

-- & C = B**(-P)
```

```

    if P = - then
        return B ;
    end if ;

    C := B ;
    for I in P + .. - loop
        C := C * B ;
    end loop ;
    return C;
end "***" ;
---
```

```

function "***" (A : VECTOR; B : VECTOR) return VECTOR is
    VTEMP : VECTOR ( .. );
    -- *****
    -- This function performs the cross product of two
    -- and B resulting in a VECTOR. Usage, C := A ** B;
    -- Comparability of dimensions is checked. LIMITED
    -- *****

begin
    if A'FIRST /= B'FIRST or A'LAST /= B'LAST then
        raise INCOMPARABLE_DIMENSION;
    end if ;
    VTEMP ( ) := A ( ) * B ( ) - A ( ) * B ( );
    VTEMP ( ) := A ( ) * B ( ) - A ( ) * B ( );
    VTEMP ( ) := A ( ) * B ( ) - A ( ) * B ( );
    return VTEMP ;
end "***";
---
```

```

function JCROSS (A : VEC T) return VEC T is
    -- *****
    -- This function rotates a Vec T degrees cw.
    -- *****
    VTEMP : VEC T;

begin
    VTEMP := (A ( ), (( - . ) * A ( )));
    return VTEMP;

```

Untitled

```
end JCROSS;

---

function JCROSS (A : MATR T) return MATR T is
  __*****
  --      This function rotates each component Vec T of M
  __*****
  B : MATR T (A'FIRST ( ) .. A'LAST ( ));

begin
  for I in A'RANGE loop
    B (I) := JCROSS (A (I));
  end loop;
  return B;
end JCROSS;

---

function ROTX (A : VEC T) return VEC T is
  __*****
  --      This function rotates a Vec T      degrees about
  __*****

begin
  return (A ( ), - A ( ));
end ROTX;

---

function ROTY (A : VEC T) return VEC T is
  __*****
  --      This function rotates a Vec T      degrees about
  __*****

begin
  return ( - A ( ), A ( ));
end ROTY;

---

function AXBDOTJ (A : VEC T; B : VEC T) return ELEMENT is
  __*****
  --Gets magnitude of A cross B for      D vectors.
```

Untitled

```

--*****
TEMP : ELEMENT;

begin
  TEMP := A ( ) * B ( ) - A ( ) * B ( );
  return TEMP;
end AXBDOTJ;
---

function GETTAN (A : VEC T; B : VEC T) return ELEMENT is
--*****
--Gets TAN(THETA) where THETA is CW angle between
--*****
EPSILON, NUM, DENOM : ELEMENT := . ;

begin
  DENOM := A * B;
  NUM := AXBDOTJ (A, B);

  if DENOM < EPSILON and DENOM >= . then
    PUT_LINE ("Tangent is beyond the limit val of");
    return (NUM / EPSILON);

  elsif DENOM > - EPSILON and DENOM < . then
    PUT_LINE ("Tangent is beyond the limit val of");
    return (( - NUM) / EPSILON);

  else
    return AXBDOTJ (A, B) / (A * B);
  end if;
end GETTAN;
end MATRIX_PACKAGE;

```