ISO/IEC JTC1/SC22/WG9 N 404
Editor's Proposal for
Revision of ISO/IEC 13813
1 March 2002

ISO/IEC JTC 1/SC 22 N

Date: 2002-03-01

ISO/IEC WD 13813

ISO/IEC JTC 1/SC 22/WG 9

Secretariat: ANSI

Generic packages of real and complex vector and matrix type declarations and basic operations for Ada

Paquetages génériques de déclarations de types de vecteur et matrice réel et complexe et opérations de base pour Ada

Document type: International Standard
Document subtype:
Document stage: (20) Preparatory
Document language: E

E:\Documents and Settings\SandoD\My Documents\ISO\ISO-IEC 13813 (E).doc STD Version 1.0

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 13813 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This second edition cancels and replaces the first edition (ISO 13813:1998), all clauses and annexes of which have been technically revised.

Introduction

The generic packages described here are intended to provide the basic real and complex vector and matrix operations from which portable, reusable applications can be built. This International Standard serves a broad class of applications with reasonable ease of use, while demanding implementations that are of high quality, capable of validation and also practical given the state of the art.

The specifications included in this International Standard are presented as compilable Ada specifications in Annex A and Annex B with explanatory text in numbered sections in the main body of text. The explanatory text is normative, with the exception of notes (labelled as such).

Generic packages of real and complex vector and matrix type declarations and basic operations for Ada

1 Scope

This International Standard defines the specifications of two generic packages of vector and matrix operations called Ada.Numerics.Generic_Real_Arrays and Ada.Numerics.Generic_Complex_Arrays. The specifications of nongeneric packages called Ada.Numerics.Real_Arrays and Ada.Numerics.Complex_Arrays are also defined, together with those of analogous packages for other precisions. This International Standard does not provide the bodies of these packages.

This International Standard specifies certain fundamental vector and matrix arithmetic operations for real and complex numbers. They were chosen because of their utility in various application areas.

This International Standard is applicable to programming environments conforming to ISO/IEC 8652:1995.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this International Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 8652:1995, *Information technology — Programming languages — Ada.*

3 Types and operations provided

The following four array types are exported by the packages provided by this International Standard:

Real_Vector	Real_Matrix
Complex_Vector	Complex_Matrix

Two composite types with elements of type Real are provided, Real_Vector and Real_Matrix, to represent real vectors and matrices, and two composite types with elements of type Complex are provided, Complex_Vector and Complex_Matrix, to represent complex vectors and matrices.

The following eighteen operations are provided:

"+"	"-"	"*"	"/"
"**"	"**abs**"	Conjugate	Transpose
Re	Im	Set_Re	Set_Im
Compose_From_Cartesian	Compose_From_Polar	Modulus	Argument
Unit_Vector	Identity_Matrix		

These are the usual mathematical operators ("+", "-", "*" and "/") for real and complex vectors and matrices (together with analogous componentwise operations for vectors); the exponentiation operator ("**") for real and complex vectors; the absolute value operator ("**abs**") for real and complex vectors and matrices; the conjugate operation (Conjugate) for complex vectors and matrices; the transpose operation (Transpose) for real and complex matrices; the cartesian component operations (Re, Im, Set_Re, Set_Im and Compose_From_Cartesian) for complex vectors and matrices, for selecting components and for composing from components; the polar component operations (Modulus, Argument and Compose_From_Polar) for complex vectors and matrices, for selecting components and for composing from components; and the initialising operations (Unit_Vector and Identity_Matrix) for real and complex vectors and matrices.

4 Instantiations

This International Standard describes generic packages Ada.Numerics.Generic_Real_Arrays and Ada.Numerics.Generic_Complex_Arrays. Each package has a generic formal parameter, which is a generic formal floating-point type named Real. At instantiation, this parameter determines the precision of the arithmetic.

This International Standard also describes nongeneric packages Ada.Numerics.Real_Arrays and Ada.Numerics.Complex_Arrays, which provide the same capability as instantiations of the packages Ada.Numerics.Generic_Real_Arrays and Ada.Numerics.Generic_Complex_Arrays. It is required that nongeneric packages be constructed for each precision of floating-point type defined in package Standard of ISO/IEC 8652:1995.

5 Implementations

An implementation of the array operations defined in Ada.Numerics.Generic_Real_Arrays and Ada.Numerics.Generic_Complex_Arrays shall conform to all of the implementation requirements specified for the corresponding (scalar) real type operations in ISO/IEC 8652:1995. An implementation of the array operations defined in Ada.Numerics.Generic_Complex_Arrays shall also conform to all of the implementation requirements specified for the corresponding (scalar) complex type operations in ISO/IEC 8652:1995.

The accuracy requirements for the results of array operations are defined in terms of corresponding accuracy requirements, specified in ISO/IEC 8652:1995, on their (real or complex) scalar elements, unless the mathematical definition of the operation includes an inner product (indicated in the specifications as such). The accuracy of operations involving inner products is beyond the scope of this International Standard, except that an implementation shall document what, if any, extended-precision accumulation of intermediate results is used to implement such inner products.

Implementations of Ada.Numerics.Generic_Complex_Arrays shall provide both a strict mode in which the accuracy requirements are observed, and an opposing relaxed mode, as defined in the Numerics Annex of ISO/IEC 8652:1995. Either mode may be the default mode, and the two modes need not actually be different. This is consistent with the numeric performance requirements for complex scalar arithmetic, and may in fact be inherited from an implementation of the package Ada.Numerics.Generic_Complex_Types specified in ISO/IEC 8652:1995.

Implementations are allowed to make reasonable assumptions about the environment in which they are to be used, but only when necessary in order to match algorithms to hardware characteristics in an economical manner. For example, an implementation is allowed to limit the precision it supports (by stating an assumed maximum value for System.Max_Digits), since portable implementations would not, in general, be possible otherwise. All such limits and assumptions shall be clearly documented. By convention, an implementation of Ada.Numerics.Generic_Real_Arrays and Ada.Numerics.Generic_Complex_Arrays is said not to conform to this International Standard in any environment in which its limits or assumptions are not satisfied, and this International Standard does not define its behaviour in that environment. In effect, this convention delimits the portability of implementations.

In implementations of Ada.Numerics.Generic_Complex_Arrays, all operations involving mixed real and complex arithmetic are required to construct the result by using real arithmetic (instead of by converting real values to complex values and then using complex arithmetic). This facilitates support for a future Ada binding to IEC 559:1989.

Some hardware and their accompanying Ada implementations have the capability of representing and discriminating between positively and negatively signed zeros as a means (for example) of preserving the sign of an infinitesimal quantity that has underflowed to zero. Implementations in which Real'Signed_Zeros is True should attempt to provide a rational treatment of the signs of zero results, result components and scalar elements of composite results.

6 Exceptions

The Constraint_Error exception, declared in package Standard of ISO/IEC 8652:1995, is raised by a subprogram in these generic packages when the argument(s) of the subprogram violate one or more of the conditions for matching elements of arrays (as in predefined equality); that is, for dyadic array operations, the bounds of the given left and right array operands need not be equal, but their appropriate vector lengths or row and/or column lengths (for matrices) shall be equal.

The Argument_Error exception, declared in package Ada.Numerics of ISO/IEC 8652:1995, is raised by a subprogram in Ada.Numerics.Generic_Complex_Arrays when the argument(s) of the subprogram violate one or more of the conditions given in the subprogram's definition.

NOTE 1 These conditions are related only to the mathematical definition of the subprogram and are therefore implementation independent.

NOTE 2 These conditions are inherited from the corresponding scalar subprogram defined in Ada.Numerics.Generic_Complex_Types of ISO/IEC 8652:1995.

An implementation shall raise the Constraint_Error exception for signalling division by zero in the following specific cases where the corresponding mathematical scalar results, or components thereof, are infinite, provided Real'Machine_Overflows is True:

a) array operations whose mathematical definition involves division of an element by (real or complex) zero;

b) array operations whose mathematical definition involves exponentiation of (real or complex) zero by a negative (integer) exponent.

If Real'Machine_Overflows is False, the result for each of the foregoing specific cases is unspecified. The Constraint_Error exception shall also be raised by a subprogram for all of the exceptional conditions related to real and complex types as defined in ISO/IEC 8652:1995, provided Real'Machine_Overflows is True.

For the case of floating-point overflow, some of the operations are allowed to raise Constraint_Error for certain arguments for which neither a result, a result component, nor a scalar element of a composite result can overflow, provided Real'Machine_Overflows is True. This freedom is granted for operations involving either an inner product or complex exponentiation. Permission to signal overflow in these cases recognizes the difficulty of avoiding overflow in the computation of intermediate results, given the current state of the art.

Besides Ada.Numerics.Argument_Error and Constraint_Error, the only exceptions allowed during a call to a subprogram in these packages are the other predefined exceptions declared in package Standard of ISO/IEC 8652:1995.

7 Generic Real Arrays Package

The generic package Ada.Numerics.Generic_Real_Arrays defines operations and types for real vector and matrix arithmetic. One generic formal parameter, the floating-point type Real, is defined for Ada.Numerics.Generic_Real_Arrays. The corresponding generic actual parameter determines the precision of the arithmetic to be used in an instantiation of this generic package.

The Ada package specification for Ada.Numerics.Generic_Real_Arrays is given in Annex A.

7.1 Types

Two types are defined and exported by Ada.Numerics.Generic_Real_Arrays. The composite type Real_Vector is provided to represent a vector with elements of type Real; it is defined as an unconstrained, one-dimensional array with an index of type Integer. The composite type Real_Matrix is provided to represent a matrix with elements of type Real; it is defined as an unconstrained, two-dimensional array with indices of type Integer.

7.2 Real_Vector arithmetic operations

```
function "+"   (Right : Real_Vector) return Real_Vector;
function "-"   (Right : Real_Vector) return Real_Vector;
function "abs" (Right : Real_Vector) return Real_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of Right. This is also the standard mathematical operation for vector identity, negation and absolute value.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652:1995.

```
function "+" (Left, Right : Real_Vector) return Real_Vector;
function "-" (Left, Right : Real_Vector) return Real_Vector;
function "*" (Left, Right : Real_Vector) return Real_Vector;
function "/" (Left, Right : Real_Vector) return Real_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of Left and the matching element of Right. This is also the standard mathematical operation for vector addition, subtraction, multiplication and division. The index range of the result is Left'Range. The exception Constraint_Error is raised if Left'Length ≠ Right'Length or if Real'Machine_Overflows is True and division by zero is attempted. If Real'Machine_Overflows is False, the result of division by zero is implementation defined.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652:1995.

```
function "**" (Left  : Real_Vector; Right : Integer) return Real_Vector;
```

This operation returns the result of applying the standard mathematical operation for exponentiation by an integer power to each element of Left. The index range of the result is Left'Range. The exception Constraint_Error is raised if for some integer I (in the index range of Left), Left(I) = 0.0 and Right < 0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of exponentiation by an integer power, as defined in ISO/IEC 8652:1995.

```
function "*" (Left, Right : Real_Vector) return Real'Base;
```

This operation returns the inner (dot) product of Left and Right.

The exception Constraint_Error is raised if Left'Length ≠ Right'Length.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

7.3 Real_Vector scaling operations

```
function "*" (Left  : Real'Base; Right : Real_Vector) return Real_Vector;
```

This operation applies the standard mathematical operation for scaling a vector Right by a real number Left. The index range of the vector result is Right'Range.

Each array element of the result shall satisfy the (scalar) accuracy requirement of multiplication, as defined in ISO/IEC 8652:1995.

```
function "*" (Left  : Real_Vector; Right : Real'Base) return Real_Vector;
function "/" (Left  : Real_Vector; Right : Real'Base) return Real_Vector;
```

Each operation applies the standard mathematical operation for scaling a vector Left by a real number Right. The index range of the vector result is Left'Range. The exception Constraint_Error is raised when division by zero is attempted and Real'Machine_Overflows is True. If Real'Machine_Overflows is False, the result of division by zero is implementation defined.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652:1995.

7.4 Other Real_Vector operations

```
function Unit_Vector (Index : Integer;
                      Order : Positive;
                      First : Integer := 1) return Real_Vector;
```

This function returns a ``unit vector'' with Order elements and a lower bound of First. All elements are set to 0.0 except for the Index element which is set to 1.0. The exception Constraint_Error is raised if Index < First, Index > First + Order - 1 or if First + Order - 1 > Integer'Last.

This function is exact.

7.5 Real_Matrix arithmetic operations

```
function "+"   (Right : Real_Matrix) return Real_Matrix;
function "-"   (Right : Real_Matrix) return Real_Matrix;
function "abs" (Right : Real_Matrix) return Real_Matrix;
```

Each operation returns the result of applying the appropriate operation to each element of Right. This is also the standard mathematical operation for matrix identity, negation and absolute value. The index ranges of the result are those of Right.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652:1995.

```
function Transpose (X : Real_Matrix) return Real_Matrix;
```

This function returns the transpose of a matrix X. The index ranges of the result are X'Range(2) and X'Range(1) (first and second index respectively).

This function is exact.

```
function "+" (Left, Right : Real_Matrix) return Real_Matrix;
function "-" (Left, Right : Real_Matrix) return Real_Matrix;
```

Each operation returns the result of applying the appropriate operation to each element of Left and the matching element of Right. This is also the standard mathematical operation for matrix addition and subtraction. The index ranges of the result are those of Left. The exception Constraint_Error is raised if Left'Length(1) ≠ Right'Length(1) or Left'Length(2) ≠ Right'Length(2).

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652:1995.

```
function "*" (Left, Right : Real_Matrix) return Real_Matrix;
```

This operation applies the standard mathematical operation for matrix multiplication. The index ranges of the result are Left'Range(1) and Right'Range(2) (first and second index respectively). The exception Constraint_Error is raised if Left'Length(2) ≠ Right'Length(1).

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

```
function "*" (Left, Right : Real_Vector) return Real_Matrix;
```

This operation applies the standard mathematical operation for multiplication of a (column) vector Left by a (row) vector Right. The index ranges of the matrix result are Left'Range and Right'Range (first and second index respectively).

Each array element of the result shall satisfy the (scalar) accuracy requirement of multiplication, as defined in ISO/IEC 8652:1995.

```
function "*" (Left : Real_Vector; Right : Real_Matrix) return Real_Vector;
```

This operation applies the standard mathematical operation for multiplication of a (row) vector Left by a matrix Right. The index range of the (row) vector result is Right'Range(2). The exception Constraint_Error is raised if Left'Length ≠ Right'Length(1).

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

```
function "*" (Left : Real_Matrix; Right : Real_Vector) return Real_Vector;
```

This operation applies the standard mathematical operation for multiplication of a matrix Left by a (column) vector Right. The index range of the (column) vector result is Left'Range(1). The exception Constraint_Error is raised if Left'Length(2) ≠ Right'Length.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

7.6 Real_Matrix scaling operations

```
function "*" (Left : Real'Base; Right : Real_Matrix) return Real_Matrix;
```

This operation applies the standard mathematical operation for scaling a matrix Right by a real number Left. The index ranges of the matrix result are those of Right.

Each array element of the result shall satisfy the (scalar) accuracy requirement of multiplication, as defined in ISO/IEC 8652:1995.

```
function "*" (Left  : Real_Matrix; Right : Real'Base) return Real_Matrix;
function "/" (Left  : Real_Matrix; Right : Real'Base) return Real_Matrix;
```

Each operation applies the standard mathematical operation for scaling a matrix Left by a real number Right. The index ranges of the matrix result are those of Left. The exception Constraint_Error is raised when division by zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652:1995.

7.7 Other Real_Matrix operations

```
function Identity_Matrix (Order            : Positive;
                          First_1, First_2 : Integer := 1) return Real_Matrix;
```

This function returns a square "identity matrix" with $Order^2$ elements and lower bounds of First_1 and First_2 (for the first and second index ranges respectively). All elements are set to 0.0 except for the main diagonal, whose elements are set to 1.0. The exception Constraint_Error is raised if First_1 + Order - 1 > Integer'Last or First_2 + Order - 1 > Integer'Last.

This function is exact.

8 Generic Complex Arrays Package

The generic package Ada.Numerics.Generic_Complex_Arrays defines operations and types for complex and mixed real and complex vector and matrix arithmetic. Two formal package parameters, Real_Arrays and Complex_Types are defined for Ada.Numerics.Generic_Complex_Arrays. The precision of the floating-point arithmetic to be used in an instantiation of this generic package is obtained from the actual parameter of the instantiation of the formal package parameters.

The Ada package specification for Ada.Numerics.Generic_Complex_Arrays is given in Annex B.

8.1 Types

Two types are defined and exported by Ada.Numerics.Generic_Complex_Arrays. The composite type Complex_Vector is provided to represent a vector with elements of type Complex; it is defined as an unconstrained, one-dimensional array with an index of type Integer. The composite type Complex_Matrix is provided to represent a matrix with elements of type Complex; it is defined as an unconstrained, two-dimensional array with indices of type Integer.

8.2 Complex_Vector selection, conversion and composition operations

```
function Re (X : Complex_Vector) return Real_Vector;
function Im (X : Complex_Vector) return Real_Vector;
```

Each function returns a vector of the specified cartesian component-parts of X. The index range of the result is X'Range.

Each function is exact.

```
procedure Set_Re (X  : in out Complex_Vector; Re : in Real_Vector);
procedure Set_Im (X  : in out Complex_Vector; Im : in Real_Vector);
```

Each procedure resets the specified (cartesian) component of each of the elements of X; the other (cartesian) component of each of the elements is unchanged. The exception Constraint_Error is raised if X'Length ≠ Re'Length and if X'Length ≠ Im'Length.

Each procedure is exact.

```
function Compose_From_Cartesian (Re     : Real_Vector) return Complex_Vector;
function Compose_From_Cartesian (Re, Im : Real_Vector) return Complex_Vector;
```

Each function constructs a vector of Complex results (in cartesian representation) formed from given vectors of cartesian component-parts (when only the real component-parts are given, imaginary component-parts of zero are assumed). The index range of the result is Re'Range. The exception Constraint_Error is raised if Re'Length ≠ Im'Length.

Each function is exact.

```
function Modulus  (X     : Complex_Vector) return Real_Vector;
function "abs"    (Right : Complex_Vector) return Real_Vector renames Modulus;
function Argument (X     : Complex_Vector) return Real_Vector;
function Argument (X     : Complex_Vector;
                   Cycle : Real'Base)      return Real_Vector;
```

Each function calculates and returns a vector of the specified polar components of X. The index range of the result is X'Range. Each array element of the result shall satisfy the (scalar) range definition of the appropriate function.

Cycle defines the period of Argument; when no Cycle is given, a period of 2π is assumed. The exception Ada.Numerics.Argument_Error is raised for Cycle ≤ 0.0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate function.

```
function Compose_From_Polar (Modulus, Argument : Real_Vector)
   return Complex_Vector;
function Compose_From_Polar (Modulus, Argument : Real_Vector; Cycle : Real'Base)
   return Complex_Vector;
```

Each function constructs a vector of Complex results (in cartesian representation) formed from given vectors of polar components. Each element of Argument is assumed to have a period of Cycle (and is reduced accordingly); when no Cycle is given, a period of 2π is assumed. The index range of the result is Modulus'Range. The exception Constraint_Error is raised if Modulus'Length ≠ Argument'Length; the exception Ada.Numerics.Argument_Error is raised for Cycle ≤ 0.0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate function.

8.3 Complex_Vector arithmetic operations

```
function "+" (Right : Complex_Vector) return Complex_Vector;
function "-" (Right : Complex_Vector) return Complex_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of Right. This is also the standard mathematical operation for vector identity and negation. The index range of the result is Right'Range.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function Conjugate (X : Complex_Vector) return Complex_Vector;
```

This function returns the result of applying the standard mathematical operation for complex conjugation to each element of X. The index range of the result is X'Range.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex conjugation.

```
function "+" (Left, Right : Complex_Vector) return Complex_Vector;
function "-" (Left, Right : Complex_Vector) return Complex_Vector;
function "*" (Left, Right : Complex_Vector) return Complex_Vector;
function "/" (Left, Right : Complex_Vector) return Complex_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of Left and the matching element of Right. This is also the standard mathematical operation for vector addition, subtraction, multiplication and division. The index range of the result is Left'Range. The exception Constraint_Error is raised if Left'Length ≠ Right'Length, and when division by (complex) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function "**" (Left : Complex_Vector; Right : Integer) return Complex_Vector;
```

This operation returns the result of applying the standard mathematical operation for complex exponentiation by an integer power to each element of Left. The index range of the result is Left'Range. The exception Constraint_Error is raised if for some integer I (in the index range of Left), Left(I) = (0.0, 0.0) and Right < 0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex exponentiation by an integer power.

```
function "*" (Left, Right : Complex_Vector) return Complex;
```

This operation returns the inner (dot) product of Left and Right; no complex conjugation is performed. The exception Constraint_Error is raised if Left'Length ≠ Right'Length.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

8.4 Mixed Real_Vector and Complex_Vector arithmetic operations

```
function "+" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "+" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Vector;
function "-" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "-" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Vector;
function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Vector;
function "/" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of Left and the matching element of Right. This is also the standard mathematical operation for vector addition, subtraction, multiplication and division. The index range of the result is Left'Range. The exception Constraint_Error is raised if Left'Length ≠ Right'Length, and when division by (real or complex) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for mixed real and complex arithmetic.

```
function "*" (Left : Real_Vector;    Right : Complex_Vector) return Complex;
function "*" (Left : Complex_Vector; Right : Real_Vector)    return Complex;
```

Each operation returns the inner (dot) product of Left and Right. The exception Constraint_Error is raised if Left'Length ≠ Right'Length.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

8.5 Complex_Vector scaling operations

```
function "*" (Left : Complex; Right : Complex_Vector) return Complex_Vector;
```

Each operation applies the standard mathematical operation for scaling a vector Right by a complex number Left. The index range of the result is Right'Range.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex multiplication.

```
function "*" (Left : Complex_Vector; Right : Complex) return Complex_Vector;
function "/" (Left : Complex_Vector; Right : Complex) return Complex_Vector;
```

Each operation applies the standard mathematical operation for scaling a vector Left by a complex number Right. The index range of the result is Left'Range. The exception Constraint_Error is raised when division by (complex) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function "*" (Left : Real'Base; Right : Complex_Vector) return Complex_Vector;
```

Each operation applies the standard mathematical operation for scaling a complex vector Right by a real number Left. The index range of the result is Right'Range.

Each array element of the result shall satisfy the (scalar) accuracy requirement of mixed real and complex multiplication.

```
function "*" (Left : Complex_Vector; Right : Real'Base) return Complex_Vector;
function "/" (Left : Complex_Vector; Right : Real'Base) return Complex_Vector;
```

Each operation applies the standard mathematical operation for scaling a complex vector Left by a real number Right. The index range of the result is Left'Range. The exception Constraint_Error is raised when division by (real) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for mixed real and complex arithmetic.

8.6 Other Complex_Vector operations

```
function Unit_Vector (Index : Integer;
                      Order : Positive;
                      First : Integer := 1) return Complex_Vector;
```

This function returns a ``unit vector'' with Order elements and a lower bound of First. All elements are set to (0.0,0.0) except for the Index element which is set to (1.0,0.0). The exception Constraint_Error is raised if Index < First, Index > First + Order - 1, or if First + Order - 1 > Integer'Last.

This function is exact.

8.7 Complex_Matrix selection, conversion and composition operations

```
function Re (X : Complex_Matrix) return Real_Matrix;
function Im (X : Complex_Matrix) return Real_Matrix;
```

Each function returns a matrix of the specified cartesian component-parts of X. The index ranges of the result are those of X.

Each function is exact.


```
procedure Set_Re (X : in out Complex_Matrix; Re : in Real_Matrix);
procedure Set_Im (X : in out Complex_Matrix; Im : in Real_Matrix);
```

Each procedure resets the specified (cartesian) component of each of the elements of X; the other (cartesian) component of each of the elements is unchanged. The exception Constraint_Error is raised if X'Length(1) ≠ Re'Length(1) or X'Length(2) ≠ Re'Length(2) and if X'Length(1) ≠ Im'Length(1) or X'Length(2) ≠ Im'Length(2).

Each procedure is exact.

```
function Compose_From_Cartesian (Re     : Real_Matrix) return Complex_Matrix;
function Compose_From_Cartesian (Re, Im : Real_Matrix) return Complex_Matrix;
```

Each function constructs a matrix of Complex results (in cartesian representation) formed from given matrices of cartesian component-parts (when only the real component-parts are given, imaginary component-parts of zero are assumed). The index ranges of the result are those of Re.

The exception Constraint_Error is raised if Re'Length(1) ≠ Im'Length(1) or Re'Length(2) ≠ Im'Length(2).

Each function is exact.

```
function Modulus  (X     : Complex_Matrix) return Real_Matrix;
function "abs"    (Right : Complex_Matrix) return Real_Matrix renames Modulus;
function Argument (X     : Complex_Matrix) return Real_Matrix;
function Argument (X     : Complex_Matrix;
                   Cycle : Real'Base)      return Real_Matrix;
```

Each function calculates and returns a matrix of the specified polar component-parts of X. The index ranges of the result are those of X. Each array element of the result shall satisfy the (scalar) range definition of the appropriate function.

Cycle defines the period of Argument; when no Cycle is given, a period of 2π is assumed. The exception Ada.Numerics.Argument_Error is raised for Cycle ≤ 0.0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate function.

```
function Compose_From_Polar (Modulus, Argument : Real_Matrix)
   return Complex_Matrix;
function Compose_From_Polar (Modulus, Argument : Real_Matrix;
                             Cycle             : Real'Base)
   return Complex_Matrix;
```

Each function constructs a matrix of Complex results (in cartesian representation) formed from given matrices of polar component-parts. Each element of Argument is assumed to have a period of Cycle (and is reduced accordingly); when no Cycle is given, a period of 2π is assumed. The index ranges of the result are those of Modulus. The exception Constraint_Error is raised if Modulus'Length(1) ≠ Argument'Length(1) or Modulus'Length(2) ≠ Argument'Length(2); the exception Ada.Numerics.Argument_Error is raised for Cycle ≤ 0.0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate function.

8.8 Complex_Matrix arithmetic operations

```
function "+" (Right : Complex_Matrix) return Complex_Matrix;
function "-" (Right : Complex_Matrix) return Complex_Matrix;
```

Each operation returns the result of applying the appropriate operation to each element of Right. This is also the standard mathematical operation for matrix identity and negation. The index ranges of the result are those of Right.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function Conjugate (X : Complex_Matrix) return Complex_Matrix;
```

This function returns the result of applying the standard mathematical operation for complex conjugation to each element of X. The index ranges of the result are those of X.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex conjugation.

```
function Transpose (X : Complex_Matrix) return Complex_Matrix;
```

This function returns the transpose of a matrix X. The index ranges of the result are X'Range(2) and X'Range(1) (first and second index respectively).

This function is exact.

```
function "+" (Left, Right : Complex_Matrix) return Complex_Matrix;
function "-" (Left, Right : Complex_Matrix) return Complex_Matrix;
```

Each operation applies the appropriate standard mathematical operation for matrix addition or subtraction. The index ranges of the result are those of Left. The exception Constraint_Error is raised if Left'Length(1) ≠ Right'Length(1) or Left'Length(2) ≠ Right'Length(2).

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function "*" (Left, Right : Complex_Matrix) return Complex_Matrix;
```

This operation applies the standard mathematical operation for matrix multiplication. The index ranges of the result are Left'Range(1) and Right'Range(2) (first and second index respectively). The exception Constraint_Error is raised if Left'Length(2) ≠ Right'Length(1).

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

```
function "*" (Left, Right : Complex_Vector) return Complex_Matrix;
```

This operation applies the standard mathematical operation for multiplication of a (column) vector by a (row) vector. The index ranges of the matrix result are Left'Range and Right'Range (first and second index respectively).

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex multiplication.

```
function "*" (Left  : Complex_Vector;
              Right : Complex_Matrix) return Complex_Vector;
```

This operation applies the standard mathematical operation for multiplication of a (row) vector by a matrix. The index range of the (row) vector result is Right'Range(2). The exception Constraint_Error is raised if Left'Length ≠ Right'Length(1).

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

```
function "*" (Left  : Complex_Matrix;
              Right : Complex_Vector) return Complex_Vector;
```

This operation applies the standard mathematical operation for multiplication of a matrix by a (column) vector. The index range of the (column) vector result is Left'Range(1). The exception Constraint_Error is raised if Left'Length(2) ≠ Right'Length.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

8.9 Mixed Real_Matrix and Complex_Matrix arithmetic operations

```
function "+" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "+" (Left  : Complex_Matrix;
              Right : Real_Matrix)    return Complex_Matrix;
function "-" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "-" (Left  : Complex_Matrix;
              Right : Real_Matrix)    return Complex_Matrix;
```

Each operation applies the appropriate standard mathematical operation for matrix addition or subtraction. The index ranges of the result are those of Left. The exception Constraint_Error is raised if Left'Length(1) ≠ Right'Length(1) or Left'Length(2) ≠ Right'Length(2).

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for mixed real and complex arithmetic.

```
function "*" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "*" (Left  : Complex_Matrix;
              Right : Real_Matrix)    return Complex_Matrix;
```

Each operation applies the standard mathematical operation for matrix multiplication. The index ranges of the result are Left'Range(1) and Right'Range(2) (first and second index respectively). The exception Constraint_Error is raised if Left'Length(2) ≠ Right'Length(1).

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

```
function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Matrix;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Matrix;
```

Each operation applies the standard mathematical operation for multiplication of a (column) vector by a (row) vector. The index ranges of the matrix result are Left'Range and Right'Range (first and second index respectively).

Each array element of the result shall satisfy the (scalar) accuracy requirement of mixed real and complex multiplication.

```
function "*" (Left  : Real_Vector;
              Right : Complex_Matrix) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real_Matrix)    return Complex_Vector;
```

Each operation applies the standard mathematical operation for multiplication of a (row) vector by a matrix. The index range of the (row) vector result is Right'Range(2). The exception Constraint_Error is raised if Left'Length ≠ Right'Length(1).

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

```
function "*" (Left  : Real_Matrix;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Matrix;
              Right : Real_Vector)    return Complex_Vector;
```

Each operation applies the standard mathematical operation for multiplication of a matrix by a (column) vector. The index range of the (column) vector result is Left'Range(1). The exception Constraint_Error is raised if Left'Length(2) ≠ Right'Length.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of Left and Right are such that computation of an intermediate result could signal overflow.

8.10 Complex_Matrix scaling operations

```
function "*" (Left : Complex; Right : Complex_Matrix) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for scaling a matrix Right by a complex number Left. The index ranges of the result are those of Right.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex multiplication.

```
function "*" (Left : Complex_Matrix; Right : Complex) return Complex_Matrix;
function "/" (Left : Complex_Matrix; Right : Complex) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for scaling a matrix Left by a complex number Right. The index ranges of the result are those of Left. The exception Constraint_Error is raised when division by (complex) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function "*" (Left : Real'Base; Right : Complex_Matrix) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for scaling a complex matrix Right by a real number Left. The index ranges of the result are those of Right.

Each array element of the result shall satisfy the (scalar) accuracy requirement of mixed real and complex multiplication.

```
function "*" (Left : Complex_Matrix; Right : Real'Base) return Complex_Matrix;
function "/" (Left : Complex_Matrix; Right : Real'Base) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for scaling a complex matrix Left by a real number Right. The index ranges of the result are those of Left. The exception Constraint_Error is raised when division by (real) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for mixed real and complex arithmetic.

8.11 Other Complex_Matrix operations

```
function Identity_Matrix (Order            : Positive;
                          First_1, First_2 : Integer := 1) return Complex_Matrix;
```

This function returns a square ``identity matrix'' with $Order^2$ elements and lower bounds of First_1 and First_2 (for the first and second index ranges respectively). All elements are set to (0.0,0.0) except for the main diagonal, whose elements are set to (1.0,0.0). The exception Constraint_Error is raised if First_1 + Order - 1 > Integer'Last or First_2 + Order - 1 > Integer'Last.

This function is exact.

9 Standard nongeneric packages

In addition to the generic packages, analogous nongeneric packages are required to define standard real and complex vector and matrix types and operations. Nongeneric packages shall be provided for all precisions defined in package Standard. The same floating-point type shall be used to generate real and complex packages of the same precision.

The packages Ada.Numerics.Real_Arrays and Ada.Numerics.Complex_Arrays shall always be provided; these packages shall define the same types and subprograms as Ada.Numerics.Generic_Real_Arrays and Ada.Numerics.Generic_Complex_Arrays, respectively, except that the predefined type Float shall replace type Real throughout.

Names of the other nongeneric packages (where defined) shall be assigned as follows:

— if the predefined floating-point type Short_Float is supported by a host implementation of ISO/IEC 8652:1995, then this type shall be used to generate the packages Ada.Numerics.Short_Real_Arrays and Ada.Numerics.Short_Complex_Arrays;

— if the predefined floating-point type Long_Float is supported by a host implementation of ISO/IEC 8652:1995, then this type shall be used to generate the packages Ada.Numerics.Long_Real_Arrays and Ada.Numerics.Long_Complex_Arrays; and

— if other predefined floating-point types are supported (e.g., Long_Long_Float), package names shall be assigned by considering the predefined types in order of ascending (for Long-types) or descending (for Short-types) precision and matching the prefix of each floating-point type with that of the corresponding package names.

Each non-generic package shall define the same types and subprograms as the corresponding generic package, except that the appropriate predefined type shall replace type Real throughout.

The nongeneric equivalent packages may, but need not, be actual instantiations of the generic package for the appropriate predefined type.

Annex A
(normative)

Ada specification for Ada.Numerics.Generic_Real_Arrays

```
generic
   type Real is digits <>;
package Ada.Numerics.Generic_Real_Arrays is

   pragma Pure(Generic_Real_Arrays);

   -- Types

   type Real_Vector is array (Integer range <>) of Real'Base;
   type Real_Matrix is array (Integer range <>, Integer range <>) of Real'Base;

   -- Subprograms for Real_Vector Types

   -- Real_Vector arithmetic operations

   function "+"   (Right : Real_Vector)       return Real_Vector;
   function "-"   (Right : Real_Vector)       return Real_Vector;
   function "abs" (Right : Real_Vector)       return Real_Vector;

   function "+"   (Left, Right : Real_Vector) return Real_Vector;
   function "-"   (Left, Right : Real_Vector) return Real_Vector;
   function "*"   (Left, Right : Real_Vector) return Real_Vector;
   function "/"   (Left, Right : Real_Vector) return Real_Vector;
   function "**"  (Left        : Real_Vector;
                   Right       : Integer)     return Real_Vector;

   function "*"   (Left, Right : Real_Vector) return Real'Base;

   -- Real_Vector scaling operations

   function "*"   (Left  : Real'Base;   Right : Real_Vector) return Real_Vector;
   function "*"   (Left  : Real_Vector; Right : Real'Base)   return Real_Vector;
   function "/"   (Left  : Real_Vector; Right : Real'Base)   return Real_Vector;

   -- Other Real_Vector operations

   function Unit_Vector (Index : Integer;
                         Order : Positive;
                         First : Integer := 1) return Real_Vector;

   -- Subprograms for Real_Matrix Types

   -- Real_Matrix arithmetic operations

   function "+"       (Right : Real_Matrix) return Real_Matrix;
   function "-"       (Right : Real_Matrix) return Real_Matrix;
   function "abs"     (Right : Real_Matrix) return Real_Matrix;
   function Transpose (X     : Real_Matrix) return Real_Matrix;

   function "+" (Left, Right : Real_Matrix) return Real_Matrix;
   function "-" (Left, Right : Real_Matrix) return Real_Matrix;
```

```
   function "*" (Left, Right : Real_Matrix) return Real_Matrix;

   function "*" (Left, Right : Real_Vector) return Real_Matrix;

   function "*" (Left  : Real_Vector; Right : Real_Matrix) return Real_Vector;
   function "*" (Left  : Real_Matrix; Right : Real_Vector) return Real_Vector;

   -- Real_Matrix scaling operations

   function "*" (Left  : Real'Base;   Right : Real_Matrix) return Real_Matrix;
   function "*" (Left  : Real_Matrix; Right : Real'Base)   return Real_Matrix;
   function "/" (Left  : Real_Matrix; Right : Real'Base)   return Real_Matrix;

   -- Other Real_Matrix operations

   function Identity_Matrix (Order            : Positive;
                             First_1, First_2 : Integer := 1) return Real_Matrix;

end Ada.Numerics.Generic_Real_Arrays;
```

Annex B
(normative)

Ada specification for Ada.Numerics.Generic_Complex_Arrays

```
generic
   with package Real_Arrays   is new Ada.Numerics.Generic_Real_Arrays   (<>);
   use Real_Arrays;
   with package Complex_Types is new Ada.Numerics.Generic_Complex_Types (Real);
   use Complex_Types;
package Ada.Numerics.Generic_Complex_Arrays is
   pragma Pure(Generic_Complex_Arrays);

   -- Types

   type Complex_Vector is array (Integer range <>) of Complex;
   type Complex_Matrix is array (Integer range <>,
                                 Integer range <>) of Complex;

   -- Subprograms for Complex_Vector types

   -- Complex_Vector selection, conversion and composition operations

   function Re (X : Complex_Vector) return Real_Vector;
   function Im (X : Complex_Vector) return Real_Vector;

   procedure Set_Re (X  : in out Complex_Vector;
                     Re : in     Real_Vector);
   procedure Set_Im (X  : in out Complex_Vector;
                     Im : in     Real_Vector);

   function Compose_From_Cartesian (Re     : Real_Vector) return Complex_Vector;
   function Compose_From_Cartesian (Re, Im : Real_Vector) return Complex_Vector;

   function Modulus  (X     : Complex_Vector) return Real_Vector;
   function "abs"    (Right : Complex_Vector) return Real_Vector renames Modulus;
   function Argument (X     : Complex_Vector) return Real_Vector;
   function Argument (X     : Complex_Vector;
                      Cycle : Real'Base)      return Real_Vector;

   function Compose_From_Polar (Modulus, Argument : Real_Vector)
      return Complex_Vector;
   function Compose_From_Polar (Modulus, Argument : Real_Vector;
                                Cycle             : Real'Base)
      return Complex_Vector;

   -- Complex_Vector arithmetic operations

   function "+"       (Right  : Complex_Vector) return Complex_Vector;
   function "-"       (Right  : Complex_Vector) return Complex_Vector;
   function Conjugate (X      : Complex_Vector) return Complex_Vector;

   function "+"  (Left, Right : Complex_Vector) return Complex_Vector;
   function "-"  (Left, Right : Complex_Vector) return Complex_Vector;
   function "*"  (Left, Right : Complex_Vector) return Complex_Vector;
   function "/"  (Left, Right : Complex_Vector) return Complex_Vector;
```

```
function "**" (Left  : Complex_Vector;
                Right : Integer)              return Complex_Vector;

function "*"  (Left, Right : Complex_Vector) return Complex;

-- Mixed Real_Vector and Complex_Vector arithmetic operations

function "+" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "+" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Vector;
function "-" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "-" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Vector;
function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Vector;
function "/" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Vector;

function "*" (Left  : Real_Vector;    Right : Complex_Vector) return Complex;
function "*" (Left  : Complex_Vector; Right : Real_Vector)    return Complex;

-- Complex_Vector scaling operations

function "*" (Left  : Complex;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Complex)        return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Complex)        return Complex_Vector;

function "*" (Left  : Real'Base;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real'Base)      return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Real'Base)      return Complex_Vector;

-- Other Complex_Vector operations

function Unit_Vector (Index : Integer;
                      Order : Positive;
                      First : Integer := 1) return Complex_Vector;

-- Subprograms for Complex_Matrix Types

-- Complex_Matrix selection, conversion and composition operations

function Re (X : Complex_Matrix) return Real_Matrix;
function Im (X : Complex_Matrix) return Real_Matrix;

procedure Set_Re (X  : in out Complex_Matrix;
                  Re : in     Real_Matrix);
procedure Set_Im (X  : in out Complex_Matrix;
                  Im : in     Real_Matrix);
```

```
function Compose_From_Cartesian (Re     : Real_Matrix) return Complex_Matrix;
function Compose_From_Cartesian (Re, Im : Real_Matrix) return Complex_Matrix;

function Modulus  (X     : Complex_Matrix) return Real_Matrix;
function "abs"    (Right : Complex_Matrix) return Real_Matrix renames Modulus;

function Argument (X     : Complex_Matrix) return Real_Matrix;
function Argument (X     : Complex_Matrix;
                   Cycle : Real'Base)      return Real_Matrix;

function Compose_From_Polar (Modulus, Argument : Real_Matrix)
   return Complex_Matrix;
function Compose_From_Polar (Modulus, Argument : Real_Matrix;
                             Cycle             : Real'Base)
   return Complex_Matrix;

-- Complex_Matrix arithmetic operations

function "+"       (Right : Complex_Matrix) return Complex_Matrix;
function "-"       (Right : Complex_Matrix) return Complex_Matrix;
function Conjugate (X     : Complex_Matrix) return Complex_Matrix;
function Transpose (X     : Complex_Matrix) return Complex_Matrix;

function "+" (Left, Right : Complex_Matrix) return Complex_Matrix;
function "-" (Left, Right : Complex_Matrix) return Complex_Matrix;
function "*" (Left, Right : Complex_Matrix) return Complex_Matrix;

function "*" (Left, Right : Complex_Vector) return Complex_Matrix;

function "*" (Left  : Complex_Vector;
              Right : Complex_Matrix) return Complex_Vector;
function "*" (Left  : Complex_Matrix;
              Right : Complex_Vector) return Complex_Vector;

-- Mixed Real_Matrix and Complex_Matrix arithmetic operations

function "+" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "+" (Left  : Complex_Matrix;
              Right : Real_Matrix)    return Complex_Matrix;
function "-" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "-" (Left  : Complex_Matrix;
              Right : Real_Matrix)    return Complex_Matrix;
function "*" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "*" (Left  : Complex_Matrix;
              Right : Real_Matrix)    return Complex_Matrix;

function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Matrix;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector)    return Complex_Matrix;
function "*" (Left  : Real_Vector;
              Right : Complex_Matrix) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real_Matrix)    return Complex_Vector;
function "*" (Left  : Real_Matrix;
              Right : Complex_Vector) return Complex_Vector;
```

```
   function "*" (Left  : Complex_Matrix;
                 Right : Real_Vector)    return Complex_Vector;

   -- Complex_Matrix scaling operations

   function "*" (Left  : Complex;
                 Right : Complex_Matrix) return Complex_Matrix;
   function "*" (Left  : Complex_Matrix;
                 Right : Complex)        return Complex_Matrix;
   function "/" (Left  : Complex_Matrix;
                 Right : Complex)        return Complex_Matrix;

   function "*" (Left  : Real'Base;
                 Right : Complex_Matrix) return Complex_Matrix;
   function "*" (Left  : Complex_Matrix;
                 Right : Real'Base)      return Complex_Matrix;
   function "/" (Left  : Complex_Matrix;
                 Right : Real'Base)      return Complex_Matrix;

   -- Other Complex_Matrix operations

   function Identity_Matrix (Order            : Positive;
                             First_1, First_2 : Integer := 1)
      return Complex_Matrix;

end Ada.Numerics.Generic_Complex_Arrays;
```

Annex C
(informative)

Rationale

C.1 Abstract

Bibliography

[1] Ada 9X Project. Ada 9X Requirements. Office of the Under Secretary of Defense for Acquisition, U.S. Department of Defense, Washington D.C., December 1990.

[2] IEEE:854 ANSI/IEEE Std 854-1987. Radix-Independent Floating-Point Arithmetic. Institute of Electrical and Electronics Engineers, New York, 1987.

[3] W. S. Brown. A Simple but Realistic Model of Floating-Point Computation, Transactions of Mathematical Software, 7:4(445--480), December 1981.

[4] R. B. K. Dewar. Proposed Ada Interface for the IEEE Standard for Binary Floating-Point Arithmetic (Version 5). Unpublished, January 1992.

[5] K. W. Dritz. Rationale for the Proposed Standard for a Generic Package of Elementary Functions for Ada. Ada Letters, XI(7):47--65, Fall 1991.

[6] G. S. Hodgson. Rationale for the Proposed Standard for Packages of Real and Complex Type Declarations and Basic Operations for Ada (including Vector and Matrix Types), Ada Letters, XI(7):131--139, Fall 1991.

[7] IEC 559:1989. Binary floating-point arithmetic for microprocessor systems. (Also: ANSI/IEEE Std 754-1985. Binary Floating-Point Arithmetic. Institute of Electrical and Electronics Engineers, New York, 1985.)

[8] ISO/IEC 8652:1987. Information technology — Programming languages — Ada (Endorsement of ANSI Standard 1815A-1983).

[9] ISO/IEC 8652:1995. Information technology — Programming languages — Ada.

[10] ISO/IEC 11430:1994. Information technology — Programming languages — Generic package of elementary functions for Ada.

[11] ISO/IEC 11729:1994. Information technology — Programming languages — Generic package of primitive functions for Ada.

[12] ISO/IEC 13814:1998. Information technology — Programming languages — Generic package of complex elementary functions for Ada.

[13] ISO/IEC/JTC1/SC22/WG9 (Ada) Ada Rapporteur Group. Ada-Comment Database. Maintained by the Ada Joint Program Office on AJPO.SEI.CMU.EDU.

[14] ISO/IEC/JTC1/SC22/WG9 (Ada) Numerics Rapporteur Group. Proposed Standard for a Generic Package of Complex Elementary Functions for Ada. Ada Letters, XI(7):140--179, Fall 1991.

[15] ISO/IEC/JTC1/SC22/WG9 (Ada) Numerics Rapporteur Group. Proposed Standard for a Generic Package of Elementary Functions for Ada. Ada Letters, XI(7):9--46, Fall 1991.

[16] ISO/IEC/JTC1/SC22/WG9 (Ada) Numerics Rapporteur Group. Proposed Standard for a Generic Package of Primitive Functions for Ada. Ada Letters, XI(7):66--90, Fall 1991.

[17] ISO/IEC/JTC1/SC22/WG9 (Ada) Numerics Rapporteur Group. Proposed Standard for Packages of Real and Complex Type Declarations and Basic Operations for Ada (including Vector and Matrix Types), Ada Letters, XI(7):91--130, Fall 1991.

[18] W. Kahan. Augmenting a Programming Language with Complex Arithmetic. UCB Report UCB/CSD 91/667, University of California at Berkeley, December 1991.

[19] J. Kok. Proposal for Standard Mathematical Packages in Ada. CWI Report NM-R8718, Centrum voor Wiskunde en Informatica, Amsterdam, November 1987.

[20] U.S. Department of Defense, Ada Joint Program Office. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A. U.S. Government Printing Office, Washington, D.C., 1983.

