Subject: Minor change in M4 to allow different-rank views of array objects
From: Van Snyder

1 Introduction

In computational applications, it is occasionally useful to be able to view an array as either a rank-one object, or as a higher rank object.

The application in which I first encountered this need was a solver for the boundary-value problem for ordinary differential equations. The solver used a method called *multiple shooting*. In this method, one divides the independent variable into several regions. One then integrates the equation from the "beginning" to the "end" of the region, while simultaneously integrating the partial derivatives of the solution values at the boundaries with respect to the solution values at the adjacent boundaries. The auxiliary equations are called *variational equations*. The variational equations are used as a Jacobian matrix – an inherently rank-two object – to solve for the state at the interior boundaries in a least-squares sense. Software to integrate the initial value problem for ordinary differential equations invariably is developed to integrate a collection of equations organized as a rank-one object. Thus, in order to form the results of integrating the variational equations into a Jacobian matrix, one needs to view part of the solution set – a rank-one object – as a rank-two object.

A more recent application arose in estimating the concentrations of several chemical species in the atmosphere, by viewed it from orbit in a direction nearly tangential to the orbit using microwave radiometer spectrometers, and scanning in a vertical direction such that the minimum perpendicular distance from the line of sight to the Earth's surface varies from 10 km to about 100 km.

The radiance observed at a given frequency in a given viewing direction can be expressed as a function of chemical species, concentration, height, orbital position, frequency, laboratory spectroscopic measurements, temperature, pressure, and numerous other factors, by a complicated integral along the line of sight, across the field of view, and in frequency. This expression is called the *radiative transfer equation*; the part of the program that evaluates the radiative transfer equation is called the *forward model*. The concentrations appear nonlinearly in the radiative transfer equation.

One inverts the radiative transfer equation by using a Newton method, to solve for the concentrations as a function of chemical species, height, and position along the spacecraft's orbit. The Newton method is an iteratvie process that starts with an estimate of the solution, uses the forward model to evaluate the radiative transfer equation and its Jacobian matrix (the derivative of the chemical concentrations with respect to the observations), then uses the Jacobian matrix to produce a refined estimate of the solution.

The forward model is written in terms of two rank-three objects – the concentration as a function of chemical species, height, and orbital position, and the measured radiance as a function of viewing direction, orbital position and frequency.

The general-purpose mathematical software package that solves the nonlinear system by applying the Newton method is written in terms of a rank-one object – the state vector – and a rank-two object – the Jacobian matrix.

Both the forward model and the Newton method package are large bodies of software. It is not feasible to re-write either one to "think" internally using the representation of the other.

It would furthermore introduce significant inefficiencies to do so. For example, the part of the Newton method software wherein most of the cycles are consumed is a linear least-squares solver based on the BLAS. One does not look forward to the possibility to re-write the BLAS to incorporate a rank-three subscript mapping, while maintaining high performance. It may at first seem that a reasonable solution is to copy the state vector between a rank-three and a rank-one representation, but in this application, it has over 100,000 elements.

In FORTRAN 77, a frequent "trick" was to keep the state vector as a rank-one object, and pass it into a procedure in which the dummy argument was declared to be a rank-three object. This subterfuge also works in Fortran 95, provided the procedure is an external procedure, and, if one provides an explicit interface for it, one lies.

2 Proposal

Malcolm has pointed out to me an equivalent but much cleaner way to do this: Allow one to create a pointer of higher rank that "views" a (part of a) rank-one object.

Work item M4 allows to specify lower bounds for pointer object dimensions during pointer assignment. The present proposal is to allow also to specify the upper bounds, so long as the target has rank one. This requires only to change the right-hand side of syntax rule R736, add a constraint, change a constraint, add a paragraph of normative text to explain the meaning of this form of pointer assignment, and add a helpful paragraph in annex C.

3 Edits

Edits refer to 00-007r2. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by immediately following text, while a page and line number followed by + (-) indicates that immediately following text is to be inserted after (before) the indicated line. Remarks for the editor are noted in the margin, or appear between [and] in the text.

R736 *bounds-spec* **is** *lower-bound* : [*upper-bound*]	136:5
Constraint: If *upper-bound* is specified for any dimension of *pointer-object* it shall be specified for all dimensions.	136:12+
[Editor: "The" ⇒ "If *upper-bound* is specified, the *target* shall have rank one; otherwise, the".]	136:17
If *upper-bound* is specified, the extent of the dimension of *pointer-object* is *upper-bound* − *lower-bound* + 1. The size of the *target* shall be the same as the size of the *pointer-object*. The *pointer-object* becomes associated with the *target*, such that the elements of *pointer-object*, taken in array element order (6.2.2.2), are the elements of the *target*.	137:14+ New ¶
[Editor: "The" ⇒ "If *upper-bound* is not specified, the".]	137:15

C.4.4$\frac{1}{2}$ Different-rank views of array objects 424:13+ New §

It is possible to obtain high-rank views of (parts of) rank-one objects, and low-rank views (in addition to sections) of parts of high-rank objects, by declaring them to be rank-one objects, and taking sections of them, or higher-rank views by specifying upper bounds in pointer assignment statements (7.5.2). For example, if one has a matrix with NR rows and NC columns, one can represent it as a rank-one object, say `A(1:NR*NC)`. One can create a rank-two view of it, say M, with a pointer assignment such as `M(1:NR,1:NC) => A`. Its super-diagonal can be accessed by `A(NR+1:NR*NC-1:NR+1)`. Its trace is `SUM(A(1::NR+1))`.