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.

I first encountered this need in a solver for the boundary-value problem for ordinary differential equations. It used a method called *multiple shooting*, in which one divides the independent variable into several regions, and then integrates the equation from the "beginning" to the "end" of the region, while simultaneously integrating the partial derivatives of the equations 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 viewing the limb from orbit using microwave radiometer spectrometers, and scanning downward from the tangent to the orbit such that the minimum perpendicular distance from the line of sight to the Earth's surface varies between about 10 and 100 km.

The radiances $\mathbf{r}$ observed at given frequencies and viewing angles below the tangent to the orbit can be expressed as a function of chemical species, concentration, viewing angle, 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, to obtain concentrations, by using an iterative process called a Newton method, that starts with an estimate of the concentrations $\mathbf{x}$, uses the forward model $\mathbf{f}$ to calculate expected radiances and the Jacobian matrix J (the derivative of the radiative transfer equation with respect to concentrations), then uses the Jacobian matrix, and the residual $\delta\mathbf{r} = \mathbf{f}(\mathbf{x}_n) - \mathbf{r}$, to produce a refined estimate of the concentrations. In simplified form, the iteration is $J^T(\mathbf{x}_{n+1} - \mathbf{x}_n) = J^T\delta\mathbf{x}_n = \delta\mathbf{r}$.

The forward model represents the concentrations $\mathbf{x}$ and the computed radiances $\mathbf{f}$ (and therefore the residual $\delta\mathbf{r}$) as rank-three objects – functions of chemical species, height, and orbital position, and of viewing angle, orbital position and frequency.

The general-purpose Newton method software package represents the concentrations $\mathbf{x}$ and the residual $\delta\mathbf{r}$ as rank-one objects.

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 concentration and residual between rank-three and a rank-one representations, but in this application, the vectors are large ($10^5 - 10^6$ elements), and the copy operations would be within the loop of the Newton iteration.

In FORTRAN 77, one could use a "trick" of keeping the state vector **x** as a rank-one object, and passing it into a procedure in which the dummy argument was declared to be a rank-three object. This works in Fortran 95, provided the procedure is an external procedure, and one lies about the interface. The danger is that a global-checking compiler will catch you lying.

2 Proposal

Malcolm has pointed out to me a cleaner way to do this: Expand the scope of work item M4 so as to allow a pointer of higher rank to "view" 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 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-bounds* are specified, the size of the *target* shall not be less than the size of the *pointer-object*. The *pointer-object* becomes associated with the *target*, such that the elements of the target of *pointer-object*, taken in its array element order (6.2.2.2), are the first SIZE(*pointer-object*) elements of the *target*. — 137:14+ New ¶

[Editor: "The" ⇒ "If *upper-bounds* are not specified, the".] — 137:15

C.4.4½ 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`. The matrix super-diagonal can be accessed by `A(NR+1:NR*NC-1:NR+1)`. Its trace is `SUM(A(1::NR+1))`.