The state_history_type module

Neil N. Carlson

July 2013
Version 1.0

Abstract

The `state_history_type` module provides a structure for maintaining the recent history of a solution procedure that is characterized by a (time) sequence of state vectors, and methods for performing polynomial interpolation based on that history.

1 Synopsis

Usage

```
use :: state_history_type
```

Derived Types

`state_history`

2 The state_history derived type

The derived type `state_history` implements a scheme for maintaining the recent history of a solution procedure characterized by a (time) sequence of state vectors, and provides a method for computing polynomial interpolation based on that history. ODE integration algorithms often require such a capability, for example. The sequence of state vectors is stored internally as a table of divided differences, which is easily updated and which makes polynomial interpolation particularly simple to express.

Object of this derived type:

- should not be used in assignment statements; only the default intrinsic assignment is available, and its semantics are unlikely to be what is desired.
- are properly finalized when the object is deallocated or otherwise ceases to exist.

The derived type has the following type bound procedures. The state vectors are implemented as rank-1 real arrays, and all real arguments and function results are of kind `real_kind()` (see the type bound function below). Currently this is `int64` from the intrinsic `iso_fortran_env` module.

2.1 Type bound subroutines

`init(mvec, {t, x [,xdot]| vlen})`
: initializes the object to maintain up to `mvec` state vectors. In the first variant, the vector `x` with time index `t` is recorded as the initial state vector of a new history. If the optional vector `xdot` is also specified, it is recorded as the state vector time derivative at the same time index. It must have the same size as `x` and the size of `x` establishes the expected size of all further state vector arguments. In the second variant, `vlen` specifies the length of the vectors to be maintained but no state vector is recorded. An object must be initialized before any of the other methods are invoked. It is permitted to re-initialize an object.

`flush(t, x [,xdot])`
: flushes the accumulated state vectors and records the state vector `x` with time index `t` as the initial state vector of a new history. If `xdot` is specified, it is also recorded as the state vector time derivative at the same time index. This differs from `init` in that the maximum number of vectors and their lengths are not changed.

`record_state(t, x [,xdot])`
: records the vector `x` with time index `t` as the most recent state vector in the history. If the vector `xdot` is present, it is recorded as the state vector time derivative at the same time index. The oldest state vector (or two oldest in the case `xdot` is present) is discarded once the history is fully populated with `mvec` vectors. Note that when only one of a `x`/`xdot` pair of vectors is discarded, it is effectively the derivative vector that gets discarded.

`get_last_state_copy(copy)`
: copies the last recorded state vector into the array `copy`, whose length should equal `state_size()`.

`get_last_state_view(view)`
: associates the array pointer `view` with the last recorded state vector. *This should be used with great caution.* The target of this pointer should never be modified or deallocated. The pointer will cease to reference the last recorded state vector when the history is subsequently modified through calls to `record_state`, `flush`, or `init`.

`interp_state(t, x [,first][,order])`
: computes the interpolated state vector at time index `t` from the set of state vectors maintained by the object, and returns the result in the user-supplied array `x`. Polynomial interpolation is used, and `order`, if present, specifies the order using the `order+1` most recent vectors; 1 for linear interpolation, 2 for quadratic, etc. It is an error to request an order for which there is insufficient data. If not specified, the maximal order is used given the available data; once the history is fully populated, the interpolation order is `mvec-1`. Typically the array `x` would have the same size as the stored state vectors, but more generally `x` may return any contiguous segment of the interpolated state starting at index `first` (default 1) and length the size of `x`.

`revise(index, x [,xdot])`
: revises the history of a selected state vector component: `index` is the component, `x`

is the new most recent value of that component, and `xdot`, if present, is the new first divided difference. All higher-order divided differences for the component are set to zero. `Depth()` must be at least 1 (2 if `xdot` is present) to use this method. The use-case for this method arises from equation switching.

2.2 Type bound functions

`real_kind()`
returns the kind parameter value expected of all real arguments.

`depth()`
returns the number of state vectors currently stored. This number will vary between 0 and `max_depth()`..

`max_depth()`
returns the maximum number of state vectors that can be stored. This number is the value of `mvec` used to initialize the object.

`state_size()`
returns the length of the state vectors stored by the object. This number will equal the size of `x` or the value of `vlen` used to initialize the object.

`last_time()`
returns the time index of the last recorded state vector.

`time_deltas()`
returns an array containing the time index differences: the first element is the difference between the last and penultimate times; the second is the difference between the last and antepenultimate times, and so forth. The length of the result equals `depth()-1`. It is an error to call this method if `depth()` is less than 2.

`defined()`
returns true if the object is well-defined; otherwise it returns false. Defined means that the data components of the object are properly and consistently defined. The function should return true at any time after the `init` method has been called; it is intended to be used in debugging situations and is used internally in assertion checks.

3 Bugs

Bug reports and improvement suggestions should be directed to `neil.n.carlson@gmail.com`