DYNAMIC VECTORS AND MATRICES Unit

Copyright *Southern Scientific cc*.
17 Capri Road
St James
South Africa
7951
November 1991

THE INTERFACE SECTION

```
const
    dynaErrmsgcount = 10;
    DynaErrMsg  : array[1..dynaerrmsgcount] of string[80]
                 =('<<<Index out of range in dynavec.put>>>',
                   '<<<Index out of range in dynavec.get>>>',
                   '<<<Negative or zero index in dynavec.expandat>>>',
                   '<<<Index out of range in dynavec.contractat>>>',
                   '<<<Index out of range in dynamat.deletecol>>>',
                   '<<<Index out of range in dynamat.deleterow>>>',
                   '<<<Row index out of range in dynamat.get>>>',
                   '<<<Row index out of range in dynamat.put>>>',
                   '<<<Index out of range in dynamat.getrow>>>',
                   '<<<Index out of range in dynamat.getcol>>>'
                  );



var
  DynaError    : integer;   {flags error conditions}

type

   pfloat    = ^float;
{=-----------------------------------}
   float     = object(tobject)      { A floating point object }
{=-----------------------------------}

      num    : double;

      constructor init(a: double);
      function    getnum: double;
      procedure   putnum( a: double);
      constructor load(var s: tstream);
      procedure   store(var s: tstream); virtual;
      end;

const

Rfloat : tstreamrec = (

      objtype  : 11500;
      vmtlink  : ofs(typeof(float)^);
      load     :@float.load;
      store    :@float.store
      );
```

```
type

    Pdynavec = ^dynavec;
{=-----------------------------------}
    dynavec  = object(tcollection)  { A simple collection of float}
{=-----------------------------------}

    constructor init(alimit,adelta : integer);
    function  get(i: integer): double;
    procedure put(i: integer; num : double);
    procedure expandat  (i : integer);
    procedure contractat(i : integer);
    function norm : double;
    end;

const

Rdynavec : tstreamrec = (

      objtype  : 11501;
      vmtlink  : ofs(typeof(dynavec)^);
      load     :@dynavec.load;
      store    :@dynavec.store
      );


type


    Pdynamat = ^dynamat;
{=-----------------------------------}
    dynaMAT = OBJECT(tobject)
{=-----------------------------------}

    nrow           : integer;
    ncol           : integer;
    rows           : pcollection;   {of dynavec }
    cols           : pcollection;   {of dynavec }

    constructor init(maxrow, maxcol : integer);
    constructor load(var s: tstream);
    procedure   store( var s: tstream);
    procedure   addrow(i : integer);
    procedure   addcol(j : integer);
    procedure   deleterow(i : integer);
    procedure   deletecol(j : integer);
    procedure   put(i,j  : integer; value : double);
    function    get(i,j  : integer) : double;
    procedure   getrow(i : integer; var pvec : pdynavec);
    procedure   getcol(j : integer; var pvec : pdynavec);
    destructor  done; virtual;

    end;

const
Rdynamat : tstreamrec = (

      objtype  : 11502;
      vmtlink  : ofs(typeof(dynamat)^);
      load     :@dynamat.load;
      store    :@dynamat.store
      );

TYPE
```

```
Pcroupier  = ^Croupier;
{---------------------------}
croupier  = object(tobject)
{---------------------------}

  index         : pdynavec;
  decksize      : integer;

  constructor init(size  : integer);
  procedure newdeck; virtual;
  function deal( deck : pcollection) : pointer; virtual;
  destructor done; virtual;
  end;



procedure printvec(var f: text;
                  width        : integer;
                  var vector   : dynavec) ;

procedure printmat(var mat: dynamat);
function  dynadotprod(a,b : dynavec) : double;
procedure printdynaerror;
```

THE FLOAT OBJECT

It may not have been wise to make an object of a floating point number, but I did this to make the streamability of the objects higher up in the hierarchy tidier. I don' t think it adds too much overhead, and the vector and matrix objects hide the Float object from the user. Besides, I like it if everything is an object. So there. But OK, screams of protest will make me change it.

The methods are really self-explanatory.

THE DYNAMIC VECTOR OBJECT

This is simply a descendant of the Tcollection object (i.e. a collection of Floats), with some extra methods to do stuff that one would want to do with vectors which can shrink and grow. The implementation sees to it that indices start at one, not zero.

constructor dynavec.init(alimit, adelta : integer);

Initializes the vector by calling the Tcollection init method, and sets all entries in the vector to zero by constructing the floats on the heap and inserting them into the collection.

procedure dynavec.put(i: integer; num : double);

Changes the value of the i'th entry to num. Posts an error in dynaerror if i doesn't make sense.

function dynavec.get(i: integer): double;

Returns the value of the i'th entry if i is OK, else posts an error in dynaerror.

procedure dynavec.expandat(i : integer);

If i positive, expands the vector at position i. Inserts a zero at index i and shifts all items one index up. Expands after the last item if i doesn't make sense.

procedure dynavec.contractat(i : integer);

Deletes and disposes the entry with index i and shifts items with higher indices down. If i doesn't make sense, does nothing and posts an error in dynaerror.

function dynavec.norm : double; {returns the norm of the object}

Returns the square root of the dotproduct of the vector with itself.

▯

THE DYNAMIC MATRIX OBJECT

The dynamat object is a desendant of tobject, and stores its entries in two collections of dynavecs, one for the rows, and one for the columns. Entries are not duplicated.

constructor dynamat.init(maxrow,maxcol : integer);

Initializes the matrix on the heap with maxcol columns and maxrow rows. All entries are zero.

constructor dynamat.load(var s : tstream);

Reads the matrix from the stream s.

procedure dynamat.store(var s : tstream);

Stores the matrix on the stream s by writing nrow, ncol and then storing the rows from row 1 to row nrow.

procedure dynamat.addrow(i : integer);

Adds a row at rowindex i. Rows with index>i shift up. If i<1 the row is added before row 1, and if i>nrow the row is added as the last row. All entries in the row are zero. The columns are adjusted to reflect the additional row.

procedure dynamat.addcol(j : integer);

Adds a column to the matrix in the same way as described for addrow.

procedure dynamat.deletecol(j : integer);

Deletes column j from the matrix, and disposes its entries. Each row is adjusted to reflect the change. If j doesn't make sense, posts an error in dynaerror.

procedure dynamat.deleterow(i: integer);

Deletes row i from the matrix in the same way as described for deletecol.

function dynamat.get(i,j : integer): double;

Returns the value at row i, column j. If the indices don't make sense, posts an error in neuralerror.

procedure dynamat.put(i,j : integer; value : double);

Sets the entry ast row i, column j to value.

procedure   dynamat.getrow(i : integer; var pvec : pdynavec);

> Pvec should be nil on entry - this routine simply sets it to point to row i. If i doesn't make sense, posts an error in dynaerror

procedure   dynamat.getcol(j : integer; var pvec : pdynavec);

> Returns with pvec pointing at column j. If j doesn't make sense, posts an error in dynaerror

destructor dynamat.done;

> Disposes the rows and their entries, then deletes all entries in the columns and disposes the columns. Calls tobject.done.

THE CROUPIER OBJECT

This was constructed in order to present training data to a neural net in random order. Some nets train better this way. The Croupier deals from a dynavec object (the deck), but does not change it.

constructor Croupier.Init(size : integer);

Initializes decksize to size and constructs an index to randomly pick from.

procedure Croupier.newdeck;

Calls index^.freeall and reconstructs index. It is the users responsibility to call this method at the correct time during a long 'dealing' session, i.e. whenever the deck is exhausted.

function Croupier.Deal(deck : pcollection) : pointer;

Returns a pointer to the randomly selected entry in the deck. The used entry in index is disposed. If index is exhausted, returns NIL.

destructor Croupier.Done;

Disposes index and calls tobject.done

```
                                    { ---- UNIT INITIALIZATION ----}

begin
    randomize;
    dynaerror  := 0;

                  {Stream Registration}

    registertype(rfloat);
    registertype(rcollection);
    registertype(rdynavec);
    registertype(rdynamat);
end.
```