MAT-file Utility Functions

Malcolm Lidierth
King's College London

These MAT-file functions can be used to manipulate large data sets in a standard version 6 MAT-file. These MAT-file utilities for writing data to a MAT-file and modifying variables mostly require that you are working in the final variable in the MAT-file i.e. the last variable saved using MATLAB's *save* function.

- **Reading data files**

A set of utilities to help with managing very large data sets that are mapped to disc are available in addition to these files (see Linkout).

▪ **where**

Where acts similarly to whos but in addition provides information about the class of the data on disc and the byte offsets into the file. This can be used to read the file using low level I/O or memmapfile

For all variables in a file

```
s=where(filename)
```

For a specified variable

```
s=where(filename, varname)
```

For a particular field of a structure

```
s=where(filename, varname, fieldname)
```

or

```
s=where(filename, varname, fieldname1, fieldname2…)
```

if you have structures within structures

Replace fieldname with propertyname for objects

`[s,swap]=where(…)` sets swap to 0, if the MAT-file endian format is the default for the platform you are using , or to 1 if byte swapping will be needed.

▪ **endian**

```
endian(filename)
```

returns 'ieee-le' for a little-endian MAT-file and 'ieee-be' for big-endian

- **Writing large data sets**

The output of where can be used to help with reading subsets of data from a variable. The following routines are to assist with writing data:

- **MATOpen**

MATOpen creates a new MAT-file, or if it exists, opens an existing MAT-file in the appropriate endian mode and returns a MATLAB file handle

```
    fh=MATOpen('myfile', permission)
```

for valid strings for permission see MATLAB's *fopen*.

All the routines below require that the target variable is the last variable in the MAT-file. This can be checked with **CheckIsLastEntry**(filename, varname) which returns true or false. If unknown, the name of the last variable in a file can be determined with **GetLastEntry**(filename) which returns a string.

In addition, all the routines require that data is stored on disc as the same class as the target variable. MATLAB's *save* command casts data to the smallest compatible data type e.g. a double array with values all between 0 and 255 will be cast to uint8. Run **RestoreDiscClass**(filename, varname) before calling the **AppendXXXX** functions to restore the class of the data on disc to that of the target variable in memory.

The AppendXXXX function can only be used with real valued variables (not complex data)

- **AppendVector**

Appends data to the end of a row or column vector

```
    AppendVector(filename, varname, vector)
```

- **AppendColumns**

Appends columns to an existing row vector or 2D matrix.

```
    AppendColumns(filename, varname, matrix)
```

Varname and matrix may be vectors or 2D matrices.

- **AppendMatrix**

Similar to AppendColumns but adds data to the final dimension of an N-dimensional matrix where N is >=2.

Suppose we have a 100x100x3 RGB image stored in variable img in myfile.mat. We can add a second 100x100x3 image from variable newimg using

```
    AppendMatrix('myfile', 'img', newimg);
```

>load myfile img
will then return a 100x100x6 matrix with the two images.
Had newimg been 100x100x6, varname would have become 100x100x9.

To organize the data into a higher dimesional matrix, use AppendMatrix in combination with AddDimension (see below)

- **AddDimension**

Can be used with AppendMatrix to save data to a higher dimension. Using the example above we could use

```
AddDimension(filename,'img'); % Convert img to a 4D matrix
                               %(100x100x3x1)
AppendMatrix(filename, 'img', newimg);% Add the 3D newimg to
the                                      %4th dimension
```

Now

```
load myfile img
```

will return a 100x100x3x2 matrix, the 4th dimension is the image number

In general, for

```
    AppendMatrix(filename, varname, matrix)
```

- If varname and matrix have the same number of dimensions, matrix will be added to the highest dimension of varname (e.g. if varname points to a 100x100x9 matrix and we add a 100x100x6 matrix, we will end up with a 100x100x15 result. The element ordering on disc will be the same as if we had *save*d a 100x100x15 matrix in the first place. MATLAB's *load* command can be used to access the matrix.
- If varname has 1 dimension more than matrix, matrix will be treated as a submatrix or (set of submatrices) and added to varname whose final dimension will be incremented by:
  size of ultimate dimension of matrix / size of penultimate dimension of varname
  which must be integer (e.g. suppose varname points to a 100x100x3x22 matrix and we add a 100x100x9 matrix, we will

produce a 100x100x3x25 result – matrix is assumed to contain three 100x100x3 matrices e.g. three RGB image frames).

Note that AddDimension adds the dimension to the data on disc. Most MATLAB functions including *load*, *whos* etc strip away any trailing singleton dimensions so the effects of AddDimension will not show until the final dimension is >=2.

- **RestoreDiscClass**

MATLAB's *save* command casts data to the smallest compatible data type e.g. a double array with values all between 0 and 255 will be cast to uint8. Run
RestoreDiscClass(filename, varname) before calling the **AppendXXXX** functions to restore the class of the data on disc to that of the target variable in memory.

- **CheckIsLastEntry & GetLastEntry**

The AppndXXX functions require that the target variable is the last variable in the MAT-file. This can be checked with **CheckIsLastEntry**(filename, varname) which returns true or false. If unknown, the name of the last variable in a file can be determined with **GetLastEntry**(filename) which returns a string.

- **VarRename**

VarRename can be used to rename a variable in a file. The original variable name can then be re-used. This is faster than using *save –append* to replace a variable.
VarRename is called in two ways:

```
bytes=VarRename(filename, varname);
```

returns the maximum length of the variable name in bytes (the number of bytes reserved on disc for the variable name)

```
res=VarRename(filename, varname, newname);
```

Replaces the variable name on disc with newname. This returns 0 if the rename has taken place, -1 otherwise. The length of newname must be less than or equal to `VarRename(filename, varname)`.