

Welcome back to **CME 292**
Advanced MATLAB for Scientific Computing
WINTER 2023

File Manipulation, Big Data Handling, & Integration with Other Languages

CME 292 LECTURE 3

1/17/2023

Outline

File Manipulation and System Interaction

- Search path
- Data import/export & file commands
- Operating system

Handling Big Data

- Efficient use of memory
- Datastore and tall arrays
- MapReduce

Integration with Other Languages

File Manipulation and System Interaction

File Manipulation, Big Data
Handling, & Integration with
Other Languages

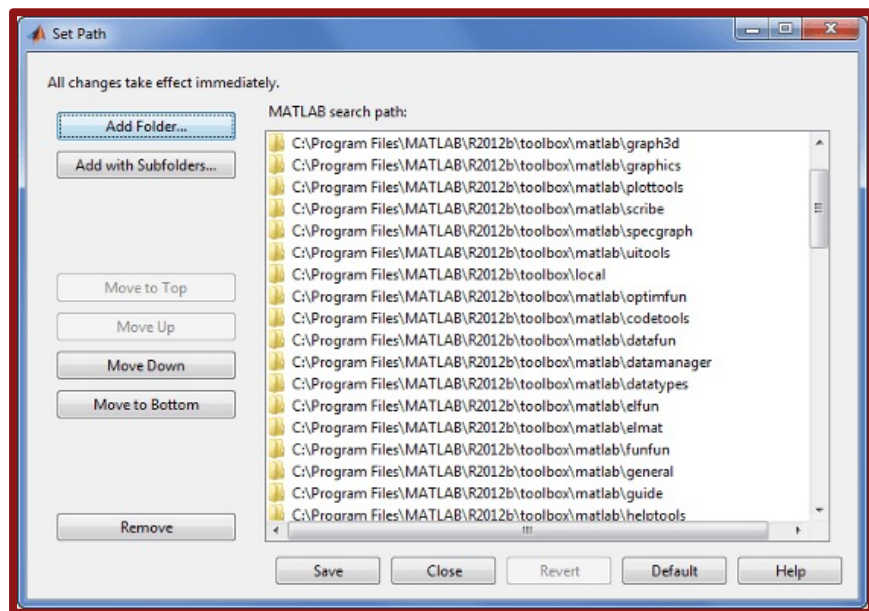


Image from MathWorks

Search Path

Search path is a subset of all folders on the file system that MATLAB uses to efficiently locate files used with MathWorks products.

- All files in the folders on search path can be accessed by MATLAB.

Order of folders on search path is important:

- When files with same filename exist in different folders, MATLAB uses the one in the folder nearest the top of the search path.

Search path includes:

- The MATLAB `userpath` folder (platform-specific)
 - added to the search path at startup
 - the default location for storing user files
- The folders defined as part of the `MATLABPATH` environment variable
- The folders provided with MATLAB and other MathWorks products, which are under `matlabroot/toolbox`.

Check Whether File or Folder on Search Path

- To determine whether a file is on the search path, run `which filename`. If the file is on the search path, MATLAB returns the full path to the file. Or,
- Use the Current Folder browser. Files and folders not on the path are dimmed.

View Entire Search Path

- Run the `path` command to view all the folders on the MATLAB search path, or,
- Use the Set Path dialog box.



Search Path Commands

- `addpath`: Add folders to search path
- `rmpath`: Remove folders from search path
- `path`: View or change search path
- `savepath`: Save current search path
- `userpath`: View or change user portion of search path
- `genpath`: Generate path string
- `pathsep`: Search path separator for current platform

Import/Export Data

Text File

- `importdata`: Load data from file
- `dlmread`: Read ASCII-delimited file of numeric data into matrix
- `dlmwrite`: Write matrix to ASCII-delimited file
- `textscan`: Read formatted data from text file or string
- `type`: Display contents of file

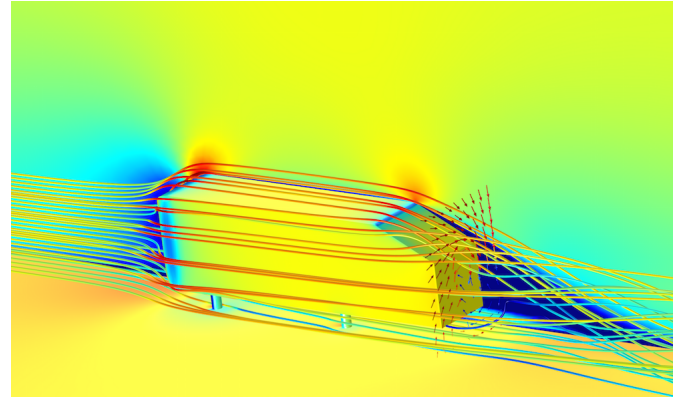
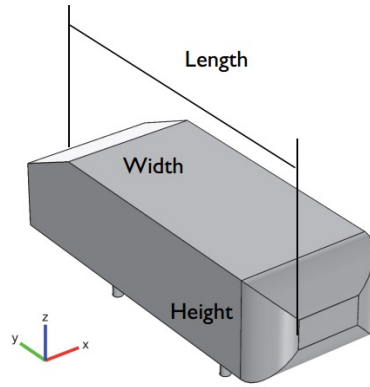
Some supported file formats for import and export

File Content	Extension	Import Function	Export Function
MATLAB formatted data	MAT	load	save
		matfile	matfile
Text	any, including: CSV, TXT	readmatrix	writematrix
		textscan	none
		readtable	writetable
		readcell	writecell
		readvars	
Spreadsheet	XLS, XLSX, XLSM, XLSB, XLTM (import only), XLTX (import only), ODS	readlines	writelines
		readmatrix	writematrix
		readtable	writetable
		readcell	writecell
		readvars	
Extensible Markup Language	XML	readstruct	writestruct
		readtable	writetable
		readtimetable	writetimetable
Image data	BMP, GIF, HDF, JPEG, JPG, JP2, JPF, PNG, TIFF, TIF, ...	imread	imwrite

Demo:

Lift-drag history for the flow around the Ahmed body

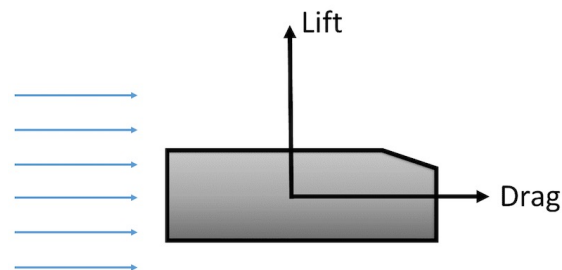
The Ahmed body is a generic car body (a simplified vehicle model).



Plots taken from the post <https://www.comsol.com/blogs/studying-the-airflow-over-a-car-using-an-ahmed-body/> and <https://www.comsol.com/blogs/how-do-i-compute-lift-and-drag/>.

Lift force & drag force:

- **Lift** is the force generated perpendicular to the direction of travel for an object moving through a fluid (gas or liquid).
- **Drag** is the force generated parallel and in opposition to the direction of travel for an object moving through a fluid. It can be broken down into the following two components:
 - Form drag (or pressure drag) - dependent on the shape of an object moving through a fluid
 - Skin friction - dependent on the viscous friction between a moving surface and a fluid, derived from the wall shear stress



CFD: computational fluid dynamics

ROM: reduced-order model

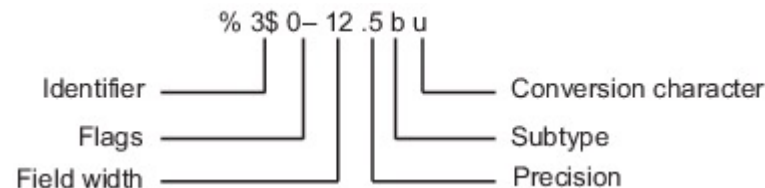
Low-Level File Commands

- `fopen`: Open file
- `fclose`: Close one or all open files
- `feof`: Test for end of file
- `ferror`: Information about file IO errors
- `fgetl`: Read line from file, remove newline character
- `fgets`: Read line from file, keep newline character
- `fileread`: Read contents of file into string
- `textscan`: Read formatted data from text file or string
- `fprintf`: Write data to text file
- `fread`: Read data from binary file
- `frewind`: Move file position indicator to beginning of open file
- `fscanf`: Read data from text file
- `fseek`: Move to specified position in file
- `ftell`: Position in open file
- `fwrite`: Write data to binary file

Format

`fprintf(FID, FORMAT, A, ...)`

Formatting operator



- Starts with a percent sign, %, and ends with a conversion character.
- Required conversion character
- Optional identifier, flags, field width, precision, and subtype
- No space between operators

Value Type	Conversion	Details
Integer, signed	%d or %i	Base 10
Integer, unsigned	%u	Base 10
	%o	Base 8 (octal)
	%x	Base 16 (hexadecimal), lowercase letters a–f
	%X	Same as %x, uppercase letters A–F
Floating-point number	%f	Fixed-point notation (Use a precision operator to specify the number of digits after the decimal point.)
	%e	Exponential notation, such as 3.1415e+00 (Use a precision operator to specify the number of digits after the decimal point.)
	%E	Same as %e, but uppercase, such as 3.1415E+00 (Use a precision operator to specify the number of digits after the decimal point.)
	%g	The more compact of %e or %f, with no trailing zeros (Use a precision operator to specify the number of significant digits.)
	%G	The more compact of %E or %f, with no trailing zeros (Use a precision operator to specify the number of significant digits.)
Characters or strings	%c	Single character
	%s	Character vector or string array. The type of the output text is the same as the type of formatSpec.

Quiz

What is the position of the end of the file

(e.g., *data/liftdrag/unsteady.hdm.LiftDrag*)?

Write a few lines of code to find this out. Consider the `fgetl` function.

Quiz

What is the position of the end of the file

(e.g., *data/liftdrag/unsteady.hdm.LiftDrag*)?

Write a few lines of code to find this out. Consider the `fgetl` function.

Solution

```
fid = fopen('data/liftdrag/unsteady.hdm.LiftDrag');  
while (fgetl(fid) ~= -1); end  
fprintf("File location = %d\n", ftell(fid));  
fprintf('At end of file? = %d\n', feof(fid));
```

Binary File vs. Text File

All files can be categorized into one of two file formats — binary or text. They encode data differently.

- While both contain data stored as a series of bits, the bits in text files represent characters, while the bits in binary files represent custom data.
- While text files contain only textual data, binary files may contain both textual and custom binary data.
- Binary files typically contain a sequence of bytes (ordered groupings of 8 bits).
- Binary files often contain headers, which are bytes of data at the beginning of a file that identifies the file's contents. Headers often include the file type and other descriptive information.
- Text files are more restrictive than binary files since they can only contain textual data.
- Comparing to text files, binary files require less storage.

Demo: (surface) mesh data in the FRG format

- First line of each file contains header information (number of nodes/elements, etc).
- Nodes contained in columns 2 - 4 for nodes file.
- Elements contained in columns 2 - 5 of elements file.

Spreadsheets

- `xlsinfo`: Determine if file contains Microsoft Excel spreadsheet
- `xlsread`: Read Microsoft Excel spreadsheet file
- `xlswrite`: Write Microsoft Excel spreadsheet file

`[STAT, MSG] = xlswrite(FNAME, M, SHEET, RANGE)`

- Writes the data in matrix M to the file FNAME in the sheet specified by SHEET to the range of cells specified by RANGE
- SHEET can be numeric specifying worksheet index or quoted string
- RANGE is of the form 'X:Y' where X indicates the upper left corner of the writable range and Y is the lower right corner (i.e. 'B2:D4' is the 3-by-3 block of cells from row B to D and columns 2 to 4)

Requires ability to use Excel as COM server; otherwise, saves to CSV file.

Workaround:

Convert data to a table and use `writetable`

`[NUM,TXT,RAW]=xlsread(FILE,SHEET,RANGE)`

- Reads the data specified in RANGE from the worksheet SHEET, in the Excel file specified in FILE.
- The full functionality of xlsread depends on the ability to start Excel as a COM server from MATLAB.

`[NUM,TXT,RAW]=xlsread(FILE,SHEET,RANGE,'basic')`

- Uses basic input mode. This is the mode used on UNIX platforms as well as on Windows when Excel is not available as a COM server.
- In this mode, xlsread does not use Excel as a COM server, which limits import ability.
- Without Excel as a COM server, RANGE will be ignored and, consequently, the whole active range of a sheet will be imported.
- Also, in basic mode, SHEET is case-sensitive and must be a string.

Image IO

- `imfinfo`: Information about graphics file
- `imread`: Read image from graphics file
- `imwrite`: Write image to graphics file

Operating System

- `clipboard`: Copy/paste strings to/from sys clipboard
- `computer`: Information about computer
- `dos`: Execute DOS command and return output
- `getenv`: Environment variable
- `perl`: Call Perl script
- `setenv`: Set environment variable
- `system`: Execute operating system command and return output
- `unix`: Execute UNIX command and return output
- `winqueryreg`: Item from Windows registry
- `bang (!)`: Shell escape

System Calls: power of operating system available inside MATLAB

- Call executable from within MATLAB
- Use MATLAB's file management to write input files and read output files
- Provide non-intrusive alternative to integrating stand-alone code with MATLAB via MEX interface

```
[status,result] = system('command')
```

- Calls upon the operating system to execute the given command. The resulting status and standard output are returned.

Handling Big Data

File Manipulation, Big Data
Handling, & Integration with
Other Languages

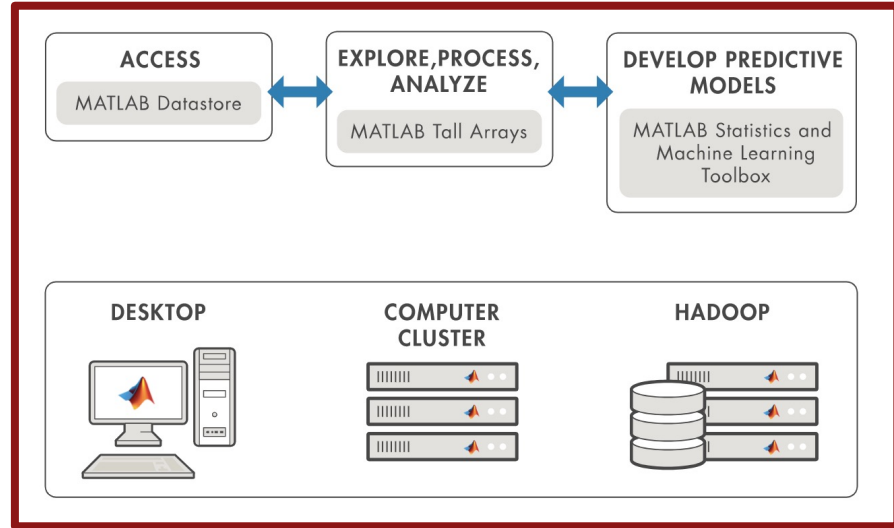
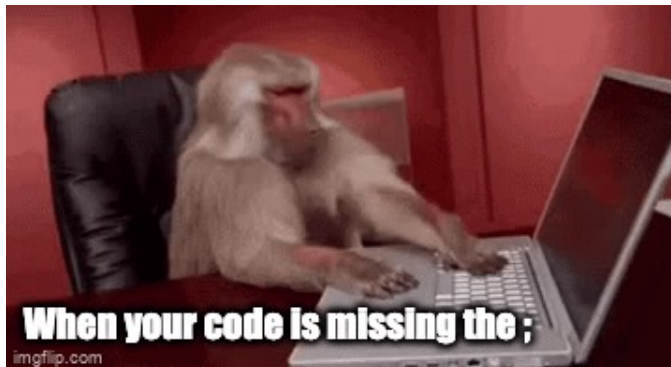


Image from MathWorks

When your variables are large in dimensions...



Efficient Use of Memory

Use Appropriate Data Storage

Data classes are of different sizes.

1. Choice of **numeric class**

- Example:
 - Default floating-point class: `double` (8 bytes)
 - Alternatives with limitations on math operations: `single` (4 bytes)
- Size of numeric classes:
 - `logical`, `int8`, `uint8` (1)
 - `int16`, `uint16` (2)
 - `int32`, `uint32` (4)
 - `int64`, `uint64` (8)

2. Overhead when storing data

- **Simple numeric arrays** (comprising one mxArray) have the least overhead, so we should use them wherever possible.
- **Cell arrays** are comprised of separate mxArrays for each element, so cell arrays with many small elements have a large overhead.
- **Structures** require a similar amount of overhead per field. Structures with many fields and small contents have a large overhead and should be avoided.
A large array of structures with numeric scalar fields requires much more memory than a structure with fields containing large numeric arrays.

3. Importation of data

- When reading data from a binary file with `fread`, it is a common error to specify only the class of the data in the file, and not the class of the data MATLAB uses once it is in the workspace.
- Example:
 - The default `double` is used even if you are reading only 8-bit values.
 - Should do: `a = fread(fid, 1e3, 'uint8=>uint8');`
 - The default for `'precision'` is `'uint8=>double'`. It means the input values are of the class specified by `uint8`. The class of the output matrix is specified by `double`.

4. Use sparse arrays whenever possible

Avoid Temporary Copies of Data

1. Avoid creating temporary arrays

- avoid creating large temporary variables
- clear temporary variables when they are no longer needed

`clear A`

2. Use nested Functions to pass fewer arguments

- When working with large data sets, be aware that MATLAB makes a temporary copy of an input variable if the called function modifies its value. This temporarily doubles the memory required to store the array, which causes MATLAB to generate an error if sufficient memory is not available.
- One way to use less memory in this situation is to use **nested functions**. A nested function shares the workspace of all outer functions, giving the nested function access to data outside of its usual scope.

Reclaim Used Memory

1. Clear large arrays that are no longer used.

2. Save large data periodically to disk

If the program generates very large amounts of data, consider writing the data to disk periodically. After saving that portion of the data, use the `clear` function to remove the variable from memory and continue with the data generation.

3. Clear old variables from memory when no longer needed

When working with a very large data set repeatedly or interactively, clear the old variable first to make space for the new variable. Otherwise, MATLAB requires temporary storage of equal size before overriding the variable.

Out of Memory

- MATLAB is a 64-bit application that runs on 64-bit operating systems. It returns an error message whenever it requests a segment of memory from the operating system that is larger than what is available.
- MATLAB has built-in protection against creating arrays that are too large. By default, MATLAB can use up to 100% of the RAM (not including virtual memory) of your computer to allocate memory for arrays, and if an array would exceed that threshold, then MATLAB returns an error.

Solutions to "Out of Memory"

1. Leverage tall Arrays

- Cast the array to a **tall array** if the large array fits in memory, but calculations run out of memory.
- Create a **datastore** from file or folder-based data, and then create a tall array on top of the datastore.

MATLAB works with small blocks of the data at a time, automatically handling all of the data chunking and processing in the background.

(More to come in next section.)

2. Leverage the Memory of Multiple Machines

- Use the Parallel Computing Toolbox™ and Distributed Arrays.

3. Load Only as Much Data as You Need

- This is a common problem with loading large flat text or binary files. The **datastore** function can help with this by allowing working with large data sets incrementally.

(More to come in next section.)

4. Increase System Swap Space

The total memory available to applications on your computer is composed of physical memory (RAM), plus a *page file*, or *swap file*, on disk. The swap file can be very large (for example, 512 terabytes on 64-bit Windows®). The operating system allocates the virtual memory for each process to physical memory or to the swap file, depending on the needs of the system and other processes. Increasing the size of the swap file can increase the total available memory, but also typically leads to slower performance.

5. Set the Process Limit on Linux Systems

The *process limit* is the maximum amount of virtual memory a single process (or application) can address. 64-bit operating systems support a process limit of 8 terabytes. On Linux systems, use `ulimit` to view and set limits including virtual memory.

6. Disable Java VM on Linux Systems

On Linux systems, starting MATLAB without the Java JVM can increase the available workspace memory by approximately 400 megabytes and increase the size of the largest contiguous memory block by about the same amount. To start MATLAB without Java JVM, use the command-line option `-nojvm`.

Datastore

A datastore is an object for reading a single file or a collection of files or data.

The datastore acts as a repository for data that has the same structure and formatting.

For example, each file in a datastore must contain data of the same type (such as numeric or text) appearing in the same order and separated by the same delimiter.

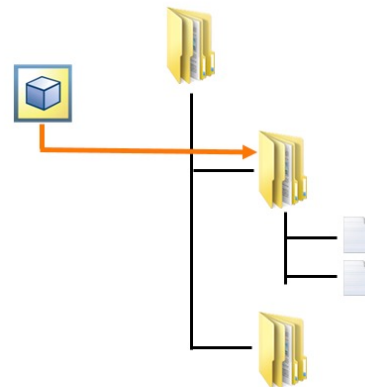
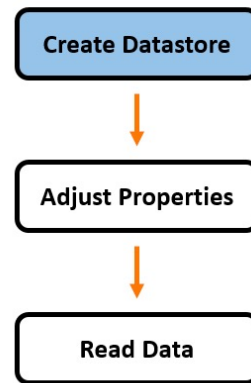
Create a Datastore

```
DS = datastore(LOCATION, 'Type', TYPE)
```

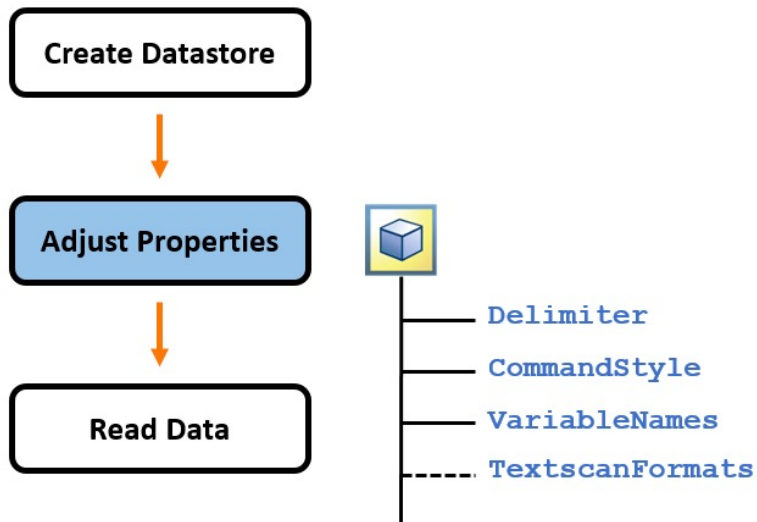
creates a datastore DS based on the LOCATION of the data and specifies the type of the datastore.

The supported types are:

- 'tabulartext' - For tabular text files
- 'image' - For image files
- 'spreadsheet' - For spreadsheet files
- 'file' - For custom format files
- 'tall' - For tall data files from tall/write
- 'keyvalue' - For use with key-value data from mapreduce
- 'database' - For use with Database Toolbox

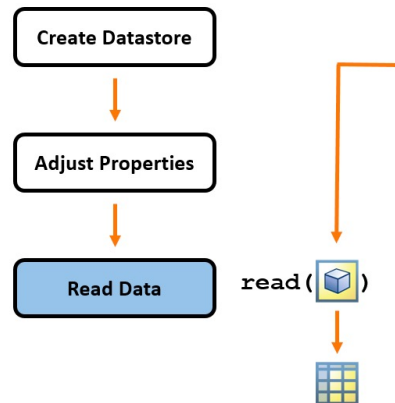


Modify datastore properties



Read from a datastore

- If all of the data in the datastore for the variables of interest fit in memory, we can read it using the `readall` function.
- `reset(ds)` resets the datastore specified by `ds` to the state where no data has been read from it. Resetting allows re-reading from the same datastore.
- When data size is large, we can choose to read the data in smaller subsets that do fit in memory via the `read` function. Change the number of rows reading in a time by assigning a new value to the `ReadSize` property.



1	Number	Timestamp	Windspeed
2	##	1 ANA M = 25 SNBR = 1192	
3	1	29-Jun-1991 12:00:00	23
4	1	29-Jun-1991 18:00:00	23
5	1	30-Jun-1991 00:00:00	23
6	1	30-Jun-1991 06:00:00	23
7	1	30-Jun-1991 12:00:00	23
8	1	30-Jun-1991 18:00:00	23
9	1	01-Jul-1991 00:00:00	23
10	1	01-Jul-1991 06:00:00	23
11	1	01-Jul-1991 12:00:00	23
12	1	01-Jul-1991 18:00:00	23
13	1	02-Jul-1991 00:00:00	23
14	1	02-Jul-1991 06:00:00	23
15	1	02-Jul-1991 12:00:00	23

Tall Array on top of the datastore

Since the data is not loaded into memory all at once, **tall arrays** can be arbitrarily large in the first dimension (having any number of rows).

Many core operators and functions work the same with tall arrays as they do with in-memory arrays.

MATLAB works with small blocks of the data at a time, handling all of the data chunking and processing in the background, so that common expressions, such as $A+B$, work with big data sets.

Benefit - deferred evaluation:

- Unlike in-memory arrays, tall arrays typically remain unevaluated until the request to perform the calculations using the **gather** function.
- MATLAB then combines the queued calculations where possible and takes the minimum number of passes through the data.
- Request output only when necessary.

MapReduce

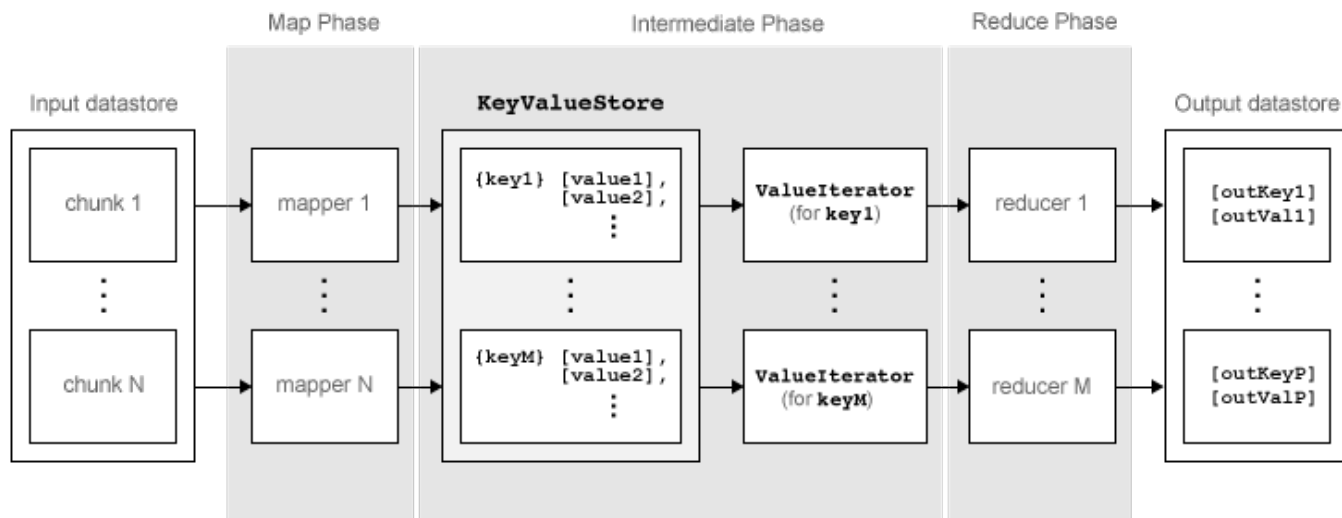
mapreduce is a programming technique which is suitable for analyzing large data sets that otherwise cannot fit in the computer's memory.

mapreduce uses a datastore to process data in small blocks that individually fit into memory.

- Each block goes through a Map phase, which formats the data to be processed.
- Then the intermediate data blocks go through a Reduce phase, which aggregates the intermediate results to produce a final result.

How MapReduce Works

- mapreduce uses a datastore to process data in small blocks that individually fit into memory.
- Each block goes through a Map phase, which formats the data to be processed.
- Then the intermediate data blocks go through a Reduce phase, which aggregates the intermediate results to produce a final result.



`outds = mapreduce(ds, mapfun, reducefun)`

- The Map and Reduce phases are encoded by *map* and *reduce* functions.
- There are endless combinations of map and reduce functions to process data.

The `mapfun` receives blocks from input datastore `ds`, and then uses the `add` and `addmulti` functions to add key-value pairs to an intermediate `KeyValueStore` object.

```
function myMapper(data, info,
    intermKVStore)
    %do a calculation with the
    data block
    add(intermKVStore, key, value)
end
```

The `reducefun` function loops through the values for each key using the `hasnext` and `getnext` functions. Then, after performing some calculation(s), it writes key-value pairs to the final output.

```
function myReducer(intermKey, intermValIter,
    outKVStore)
    while hasnext(intermValIter)
        X = getnext(intermValIter);
        %do a calculation with the current value, X
    end
    add(outKVStore, key, value)
end
```

Database Toolbox

The Database Toolbox provides functions and an app for exchanging data with relational and nonrelational databases. It enables this exchange by automatically converting between database and MATLAB data types.

E.g.,

- Import data from and export data to **relational databases** directly within MATLAB. Use knowledge of SQL to exchange data programmatically, or interact with data without using SQL.
- Use the MATLAB Interface to **SQLite** to create, read, and write relational data from SQLite database files without installing or administering database software or drivers.
- Access and import columnar data using the Database Toolbox interface for the **Apache Cassandra** database.
- Store, query, and retrieve unstructured and structured data using the Database Toolbox interface for **MongoDB**.
- Explore, manage, and store graph data in **Neo4j** databases using the MATLAB interface to Neo4j. Perform graph network analysis with or without existing knowledge of the Cypher® graph query language.

Integration With Other Languages

File Manipulation, Big Data Handling, & Integration with Other Languages



Image from MathWorks

Calling Python from MATLAB

Directly call Python functionality from MATLAB

Add the **py.** prefix to the Python name to access Python libraries.

Python standard library: **py.** in front of the Python function or class name

Available modules: **py.** in front of the Python module name followed by the Python function or class name.

Current environment of Python interpreter:

pyenv

Calling MATLAB from Python

Install MATLAB Engine API for Python

https://www.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html

```
cd (fullfile(matlabroot,'extern','engines','python'))  
system('python setup.py install')
```

- Start Python, import the package :
`import matlab.engine`
- Start a new MATLAB process
`eng = matlab.engine.start_matlab()`
- Stop Engine
`eng.quit()`
- Call MATLAB Functions from Python
`tf = eng.isprime(37)`
`print(tf)`
- Return Multiple Output Arguments
`t = eng.gcd(100.0, 80.0, nargout=3)`
`print(t)`

Calling C/C++ from MATLAB

Get source code example `arrayProduct.cpp`.

This multiplies an array by a scalar input and returns the resulting array.

```
copyfile([matlabroot
'/extern/examples/cpp_mex/arrayProduct.cpp'], '.', 'f')
```

You may edit the file by

```
edit([matlabroot
'/extern/examples/cpp_mex/arrayProduct.cpp']);
```

Take a look at the `.cpp` file.

The usage of the file is

```
outMatrix = arrayProduct(multiplier, inMatrix)
```

Build the MEX file by

```
mex arrayProduct.cpp
```

Calling MATLAB from C/C++

Use the `feval` and `fevalAsync` member functions of the `matlab::engine::MATLABEngine` class to evaluate MATLAB statements using variables in the MATLAB base workspace.

```
#include "MatlabEngine.hpp"
```

```
// Start MATLAB engine synchronously  
std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();
```

```
// Call MATLAB function and return result  
matlab::data::Array results = matlabPtr->feval(u"func", args);
```


Next Lecture

Statistical and Machine Learning

- ML refresher
- Data preprocessing & feature extraction
- Unsupervised learning
- Classification
- Regression

Deep Learning

- DL refresher
- Visualization methods
- Use pretrained model
- Create a network

Fun with MATLAB

Type `life` in the Command Window.

Type `lorenz` in Command Window.