Scientific Computing Lecture 3: Advanced data types and solving ODEs

InFoMM CDT
Mathematical Institute
University of Oxford

Andrew Thompson
(slides by Stuart Murray)

Some more data types

- A text **string** can be entered by enclosing text within single quotes, and has the char data type.

```
>>str = 'MATLAB is awesome!';
```

Some more data types

- A text **string** can be entered by enclosing text within single quotes, and has the char data type.
  ```
  >>str = 'MATLAB is awesome!';
  ```
- **Cell arrays** are collections of data of varying types, and are entered using curly brackets.
  ```
  >>mycell = {'Freedom!',randn(3,3)};
  ```

Some more data types

- A text **string** can be entered by enclosing text within single quotes, and has the char data type.
  ```
  >>str = 'MATLAB is awesome!';
  ```
- **Cell arrays** are collections of data of varying types, and are entered using curly brackets.
  ```
  >>mycell = {'Freedom!',randn(3,3)};
  ```
- **Structures:** A struct is a variable with several parts.
  ```
  >>andrew.name = 'Andrew Thompson';
  >>andrew.email = 'thompson@maths.ox.ac.uk';
  >>andrew.favouritenumber = 42;
  ```
- ```
  >>andrew
             name:  'Andrew Thompson'
            email:  'thompson@maths.ox.ac.uk'
   favouritenumber:  42
  ```
  ```

Sparse matrices

Memory

In many problems we deal with matrices that are sparse (most entries are zero)

Arrays containing many zeros waste memory in MATLAB

MATLAB can store sparse matrices in a special way:

Only nonzero elements and their positions are stored

All other entries are taken to be zero

We use `sparse` to create a sparse matrix:

```
A = diag(1:10000);      % create a diagonal matrix
S = sparse(A);
```

Convert from a sparse matrix to a full matrix using `full`

Take a look at A and S in memory using whos:

```
>> whos
    Name        Size              Bytes     Class     Attributes
    A           1000x1000     800000000     double
    S           1000x1000        160004     double    sparse
```

Memory requirement is reduced to around 1/10,000 of that for A

MATLAB remembers that S is a sparse matrix

All of MATLAB's built-in arithmetic, logical and indexing operations work with sparse matrices.

Operations with sparse matrices will return sparse matrices.

Speed

Let us compare some operations using the timers `tic` and `toc`:

```
tic
A+A;
toc


tic
S+S;
toc
```

We get the following results

```
Elapsed time is 1.250672 seconds.
Elapsed time is 0.109196 seconds.
```

The expression `A^2` even causes my machine to run out of memory, while `S^2` completes quickly

ODE solvers

Why so many?

MATLAB has many built-in functions for numerically solving ODEs

Here is a partial list:

ode45	Medium accuracy solver: first port of call for all problems
ode23	For solving systems with crude error tolerance
ode113	For systems with stringent error tolerance
ode15s	Stiff system solver (because ode45 has proved too slow)

We will consider only ode45

It is a very good general purpose routine, and usually efficient/accurate enough

ODE solving

We use ode45 to solve a system of ODEs of the form

$$\begin{cases} y_1' = f_1(t, y_{1,} y_{2,} .., y_n) \\ y_2' = f_2(t, y_{1,} y_{2,} .., y_n) \\ y_3' = f_3(t, y_{1,} y_{2,} .., y_n) \\ \vdots \\ y_n' = f_n(t, y_{1,} y_{2,} .., y_n) \end{cases}$$

i.e. $\mathbf{y}' = f(t, \mathbf{y})$

We supply ode45 with three arguments:

a handle to a function to compute the right-hand sides

a vector of start and stop times

a vector of initial conditions for each y

Computing the right-hand sides

We write a function that given values $\mathbf{y}$ and t, returns the right-hand side

Example: solve $y'=y, \quad y(t=0)=1$:

```
function [dy] = myFun(t,y)
    dy = y;
end
```

An example of a system: solve the equations

$$\begin{cases} y_1'=y_2 \\ y_2'=\sin(y_1), \quad y_1(t=0)=y_2(t=0)=1 \end{cases}$$

```
function [dy] = myFun2(t,y)
    dy = zeros(2,1)     % make a column vector
    dy(1) = y(2);
    dy(2) = sin(y(1));
end
```

Calling ode45

We call ode45 using a function handle like this:

```
sol = ode45(@myFun,[0 100], [1])
```

output handle time range i.c.s

For our second example, the call would look like this:

```
sol = ode45(@myFun2,[0 10], [1 1])
```

The function deals with everything including the time stepping

Solution information is stored in `sol`

If you omit the left-hand variable the ode45 produces plots showing the solutions automatically

Example solutions