MACSI One Day Graduate Course:
Numerical Solution to Differential Equations using Matlab

Part 2: Matlab Implementation

Niall Madden and Meghan Stephens
Department of Mathematics

National University of Ireland, Galway
Ollscoil na hÉireann, Gaillimh

3rd of April 2007.

- **Matlab** is an interactive environment for mathematical and scientific computing. It the standard tool for numerical computing in industry and research.

- **Matlab** is an interactive environment for mathematical and scientific computing. It the standard tool for numerical computing in industry and research.
- Matlab stands for Matrix Laboratory. It specialises in Matrix and vector computations, but includes functions for graphics, numerical integration and differentiation, solving differential equations, etc.

- **Matlab** is an interactive environment for mathematical and scientific computing. It the standard tool for numerical computing in industry and research.
- Matlab stands for Matrix Laboratory. It specialises in Matrix and vector computations, but includes functions for graphics, numerical integration and differentiation, solving differential equations, etc.
- Matlab differs from most significantly from, say, Maple, by not having a facility for abstract computation (although there is a toolbox than can interface with Maple).

Matlab Basics

Matlab an interpretive environment – you type a command and it will execute it immediately.

The default data-type is a matrix of double precision floating-point numbers. A scalar variable is in instance of a 1×1 matrix. To check this set, say, `t=10`, and use the `size(t)` command to find the numbers of rows and columns of t.

Matlab Tip

Type
```
>> who
```
or
```
>> whos
```
to list all your defined variables.

Matlab Basics

A vector may be declared as follows:
```
x = [0 0.25 0.5 0.7 1.0];
```
This generates a vector `x` with $x_1 = 0$, $x_2 = 0.25$, etc.

This is the same as
```
x = [0, 0.25, 0.5, 0.75, 1.0];
```
and represents a ***row*** vector.

To access a member of a vector, type
```
>> x(3)
```

Two other ways of defining this vector:

```
x = [0, 0.25, 0.5, 0.75, 1.0]
```
is the same as
```
x = 0:0.25:1
```
and
```
x = linspace(0, 1, 5)
```


Matlab Basics

Matlab Tip

Matlab indexes all vectors starting at `1`. So, for example, we think of the finite difference mesh as

$$\{x_0, x_1, \ldots, x_n\}$$

we programme it as

```
x(1), x(2), x(3), ..., x(n+1)
```


Matlab Basics

The above examples construct ***row*** vectors. It is more natural to work with ***column*** vectors.

```
x = [0 ; 0.25 ; 0.5 ; 0.75 ; 1.0]
```

Since row vectors are the default, we might need to get the transpose of a vector:
```
x = (0:0.25:1)'
```

Matlab Basics

To define a Matrix:
`A = [3, 1, 0; 1, 3, 1; 0, 1, 3];`

To access the entry in row i, column j:
`A(i,j)`

Important Matrix Operations

- Eigenvalues: `eigs(A)`
- Determinant: `det(A)`
- Norm: `norm(A)`
- Condition number: `cond(A)`
- the Inverse: `inv(A)`

Also, addition and multiplication is just done the obvious way using the `+` and `*` operators.

Matlab Basics

Most "scalar" functions return a matrix when given a matrix as an argument. For example, if x is a vector, then
`y = sin(x)`
sets y to be a vector such that $y_i = \sin(x_i)$.

Matlab has most of the standard mathematical functions: `sqrt`, `sin`, `cos`, `exp`, `log`, etc.
In each case, write the function name followed by the argument in round brackets, e.g.,
`exp(x)` for e^x.

Matlab Basics

As mentioned, the `*` operator performs matrix multiplication.
For element-by-element multiplication use `.*`

For example,
`y = x.*x`
sets $y_i = (x_i)^2$. So does `y = x.^2`

Similarly, `y=1./x` set $y_i = 1/x_i$.

Matlab Basics

If you put a semicolon at the end of a line of Matlab, the line is executed, but the output is not shown.

This is useful if you are dealing with large vectors.

If no semicolon is used, the output is shown in the command window.

Plotting functions

Define a vector
`x = [0:0.25:1]'`,
and then set
`f = x + sqrt(x) -1`
To plot these vectors use:
`plot(x, f)`
If the picture isn't particularly impressive, then this might be because Matlab is actually only printing the 4 points that you defined. To make this more clear, use
`plot(x, f, '-o')`
This means to plot the vector f as a function of the vector x, placing a circle at each point, and joining adjacent points with a straight line.

Script files

When programming in Matlab, we add the Matlab commands to a `dot m` file and execute that by typing its name.

Open a file, and add the lines:

```
%%% ScriptExample.m
%%% First Script file for Matlab Workshop
%%% 03/04/2007

clear %% Get rid of all declared variables.

N = 4
x = linspace(0,1,N+1)';
f = x + sqrt(x) -1;

plot(x, f, '-o');
```


`inline` functions

We want to write our own functions.

Some will be quite complicated:
`u = FiniteDifference(r, f, N);` In this case, we'll write them as a **`Matlab Function file`**.

But others functions, for example the coefficients in our differential equation are relatively simple, e.g., $r(x) = e^x + x$.

In this case, we can define ***inline*** functions:

`r = inline('exp(x) + x')`

`for` loops

Syntax:

```
for i = array
   .
   .
   .
end
```
where *array* is a vector.

At each iteration of the loop, i takes on successive values of the vector.

Recall our differential equation is

$$-u''(x) + r(x)u(x) = f(x) \quad \text{for } 0 < x < 1.$$
$$u(0) = \alpha, u(1) = \beta.$$

Our method is:

- Choose N, the number of *mesh intervals*
- Set up a set of $N+1$ equally spaced points:

$$0 = x_0 < x_1 < x_2 < x_3 \cdots < x_{n-1} < x_n = 1.$$

- Construct A, a $(N+1) \times (N+1)$ matrix of zeros, except for
  - $A_{1,1} = 1$
  - For $k = 2, 3, \ldots, N$

$$A_{k,k-1} = -\frac{1}{h^2}, \qquad A_{k,k} = \frac{2}{h^2} + r_k \qquad A_{k,k+1} = -\frac{1}{h^2}$$

  - $A_{N,N+1} = 1$

```
A(1,1) = 1;
for i=2:N
    A(i,i-1) = -1/h^2;
    A(i,i)   = 2/h^2 + r(x(i));
    A(i,i+1) = -1/h^2;
end
A(N+1,N+1)=1;
```


Important

We would **never** do this in practise.

Most high-level languages require that variables are declared before being used. For arrays, the dimension must be given.

In Matlab, this is not the case.

The default data type is `double` and the system dynamically re-sizes vectors and matrices as required.

However this is very slow.

We've already seen that we can find the inverse of a Matrix using the **`inv`** function.

So we could solve the system

$$A\mathbf{u} = \mathbf{f}$$

by

```
u = inv(A)*f
```

But in practise we never do. Instead use

```
u = A \ f
```

This is the Matlab **`mldivide`** (Matrix Left Divide) function and solves the system using a form of Gaussian Elimination

Solving large systems of equations is a deep and important topic, but beyond the scope of this workshop.

Putting it all together

```
%%% FiniteDifference.m
%%% Date: 03/04/07
%%% Author: Niall Madden,NUI Galway
%%%
%%% This is a simple *script* file that implements a
%%% finite difference method for the problem
%%% -u''(x) + r(x) u(x) = f(x) on (0,1)
%%% u(0)=alpha, u(1) = beta,  on a uniform mesh
clear;

%% Problem Data
alpha = 0; beta = 0;
r = inline('4+x*0');
f = inline('1+exp(x)');
%%
```


Putting it all together

```
%% Data for numerical method
N = 8;
%%

%% Set up the mesh
h = 1/N;
x = linspace(0, 1, N+1);
```


Putting it all together

```
%% Construct the linear system
A(1,1) = 1;
F(1) = alpha;
for k=2:N
    A(k,k-1) = -1/h^2;
    A(k,k)   = 2/h^2 + r(x(k));
    A(k,k+1) = -1/h^2;

    F(k) = f(x(k));
end
A(N+1,N+1)=1;
F(N+1) = beta;
```


Putting it all together

```
%% Solve the linear system
U = A\F';

%% Plot the output
plot(x, U, '-o');
```

The Profiler

As mentioned above, this is not a good way to construct a linear system.

Whenever you write a Matlab program, particularly for solving differential equation, you should use the profiler to find any bottle-necks in the code.

If most of the time is **not** spent solving the linear system, then there is a problem.

Another simple method for code-timing are the `tic` and `toc` functions.

Some Optimisations

The improve, and speed up this code, initialise the matrix `A` and vector `F`:

```
A = zeros(N+1, N+1);    F = zeros(N+1,1)
```

However, the real improvement is:

DO NOT USE `for` LOOPS TO INITIALISE MATRICES OR VECTORS.

For vectors, this is easy:

```
F = [alpha; r(x(2:N)); beta];
```

For Matrices, this requires a little more care. Also, its useful at this stage to introduce the idea of a ***sparse*** matrix.

Sparse Matrices

A Matrix is ***sparse*** if it makes sense to store only the non-zeros entries.

Matlab has a sparse matrix type the hides (most of the) details from the user/programmer.

To initialise:

```
A = sparse(N+1, N+1);
```

However, the best way to use it is as: `S = sparse(i,j,s)` which sets `S(i(k),j(k)) = s(k)`

Sparse Matrices

```
%% Construct the Linear System
Main_Diagonal = [1; 2/h^2+r(x(2:N)); 1];
Sup_Diagonal  = -ones(N-1,1)/h^2;
Sub_Diagonal  = -ones(N-1,1)/h^2;

A = sparse([1:N+1, 2:N,   2:N], ...
           [1:N+1, 3:N+1, 1:N-1], ...
     [Main_Diagonal', Sup_Diagonal', Sub_Diagonal']);
```

Matlab Functions

The script file that we have developed so far will compute an approximate solution to our differential equation.

We should now *test* this to verify that the computed results are reasonable.

But rather than employing a script file, we'll re-frame it as a function file.

This is done by writing an `m`-file that begins with the keyword `function`, followed by

- return value
- =
- function (file) name
- arguments

Example

```
function U = Solve_BVP(r, f, alpha, beta, x)
```