Introduction to Matlab

Roger Hansen (rh@fys.uio.no)

PGP, University of Oslo

September 2004

Contents

- Programming Philosophy
- What is Matlab?
- Example: Linear algebra
- Example: Curve fitting
- Plotting
- Programming techniques
- Example: Diffusion equation

Programming Philosophy

There are many different programming languages made for different purposes.

High level Offers advanced built-in functions. Examples are Matlab and Perl. Easier to learn and program.

Low level More focus on details. Examples are C and Fortran. Programming is more tedious and error-prone.

We also distinguish between general purpose (C) and special purpose languages (Matlab).

What is Matlab?

Matlab is a tool for doing scientific computing and visualization.

- Interactive
- Lots of built-in functions
- Advanced plotting features
- (Relatively) easy to do advanced computation and visualization.

It is not the holy grail. Some problems are not easily solved with Matlab.

Example: Matrix system

Matlab has a wide set of operators and functions for calculations with matrices and vectors.

We start by solving a system of linear equations

$$\begin{aligned} 3x_1 &+ 2x_2 &+ x_3 &= 39 \\ 2x_1 &+ 3x_2 &+ x_3 &= 34 \\ x_1 &+ 2x_2 &+ 3x_3 &= 26 \end{aligned} \tag{1}$$

Using matrix notation the system looks like

$$Ax = b, \tag{2}$$

where

$$A = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \end{bmatrix} x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} b = \begin{bmatrix} 39 \\ 34 \\ 26 \end{bmatrix} \tag{3}$$

The solution to the system is given by

$$x = A^{-1} * b \tag{4}$$

In Matlab we can solve this problem in two ways.

1. Find x by left-multiplication of b with A^{-1}
2. Left-division of b with A

The matlab code looks like

```
A = [3, 2, 1; 2, 3,1; 1, 2, 3];
b = [39; 34; 26];
x = inv(A)*b
```

In the second case left-division is performed straight away with the command

```
x = A\b
```

The latter is fastest.

Efficiency

It is easy to verify that second case is fastest.

```
A = rand(1000,1000); % Creates random matrix A
b = rand(1000,1);    % Creates random vector b

det(A)               % Calculates the
                     % determinant  of A

tic,                 % Starts the time-watch
x = inv(A)*b;        % Solves the system
toc                  % Stops the watch

tic, y = A\b; toc    % Solves the system
                     % with left division
```

Example: Curve Fitting

A dataset may be fitted into a polynomial of degree N by use of the function polyfit. The error is minimized in a least square-sense.
A = polyfit(y,x,N)
returns the coefficients of the polynomial $y_p(x)$ defined as

$$y_p(x) = a_N x^N + a_{n-1} x^{N-1} + \cdots + a_2 x^2 + a_1 x + a_0 \qquad (5)$$

where $A = [a_n, a_{N-1}, \cdots, a_2, a_1, a_0]$

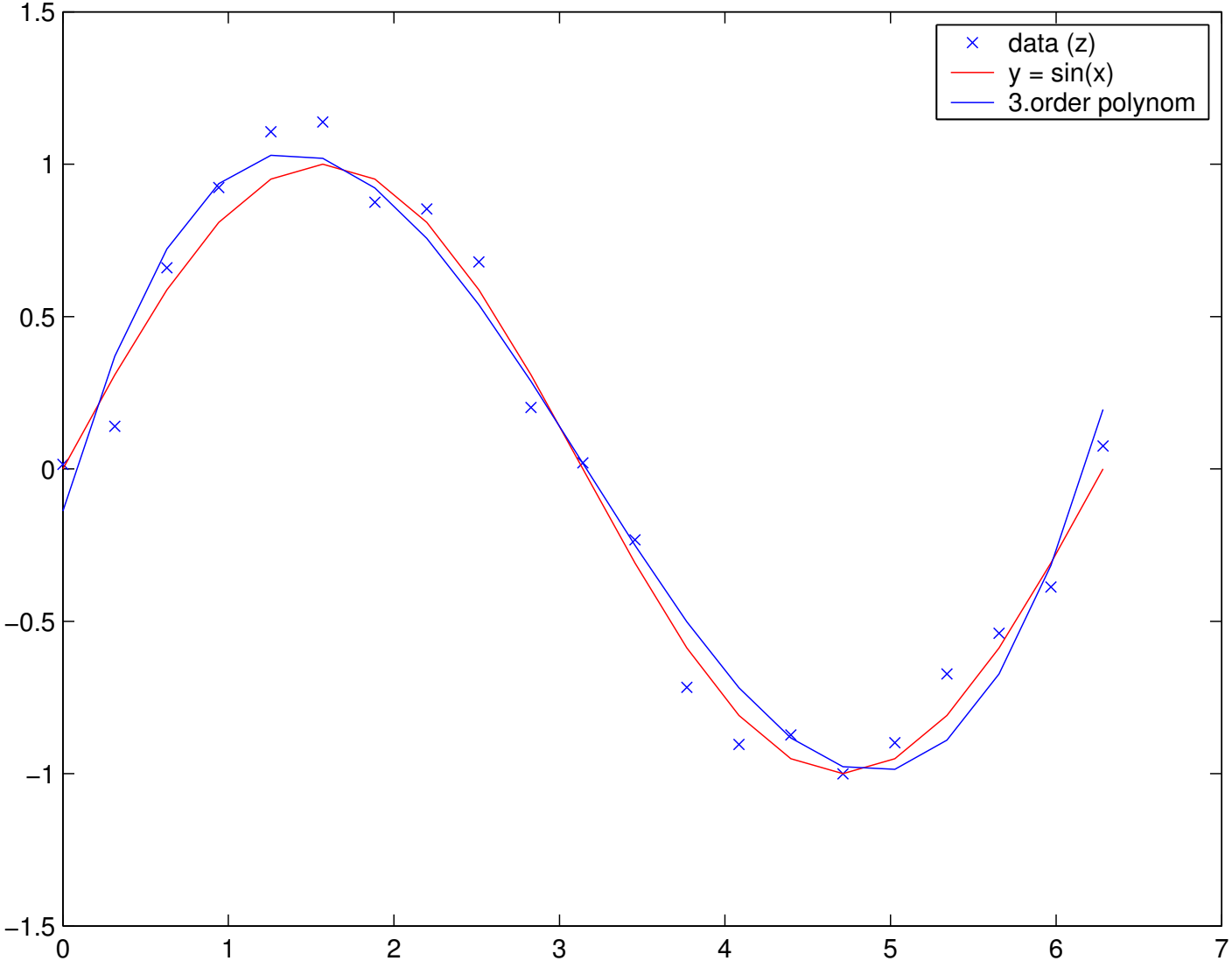
The next slides shows some examples.

Here we create a data set that almost match the sine function. Then we plot it with the sine function and a 3.order polynomial function.

```
x = 0:pi/10:2*pi;     % Create array x
y = sin(x);           % Sine of x
a = randn(1,21)*0.1; % array of random numbers
                      % with normal distribution
z = y+a;

plot(x,z,'x', x,y,'r');
hold on;              % show plots in same window

p3 = polyfit(x,z,3); % Create polynom of degree
f3 = polyval(p3,x);
plot(x,f3,'b');
legend('data (z)','y = sin(x)',
       '3.order polynom');
```

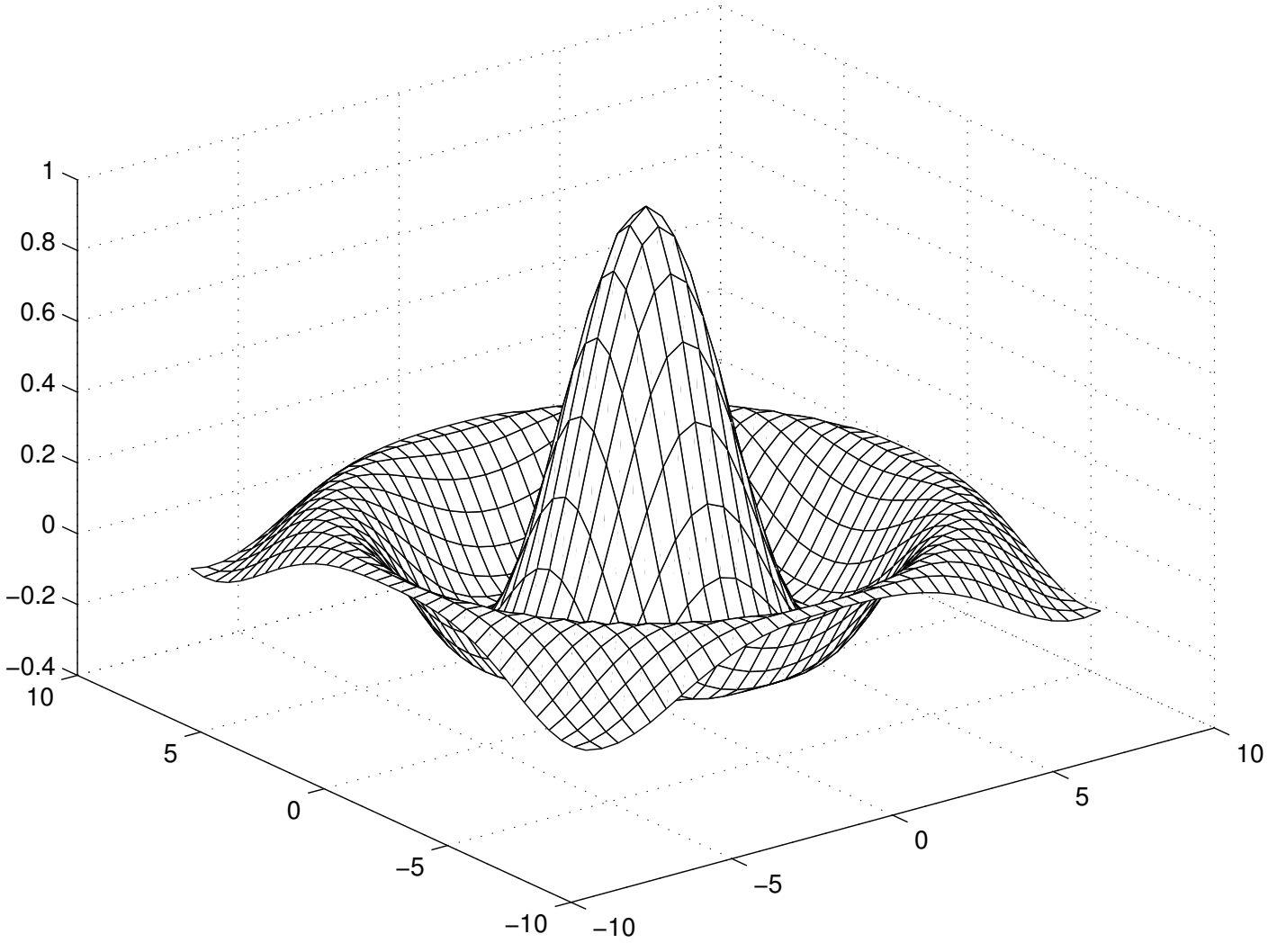

data (z)
y = sin(x)
3.order polynom
1.5
1
0.5
0
−0.5
−1
−1.5
0
1
2
3
4
5
6
7

More Plotting

The last example shows plotting of 1D arrays representing discrete data and functions. Matlab can of course also do 2D and 3D plotting.
The next example shows how to make a mesh plot.

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(X,Y,Z,'EdgeColor','black')
```

Mesh Plot

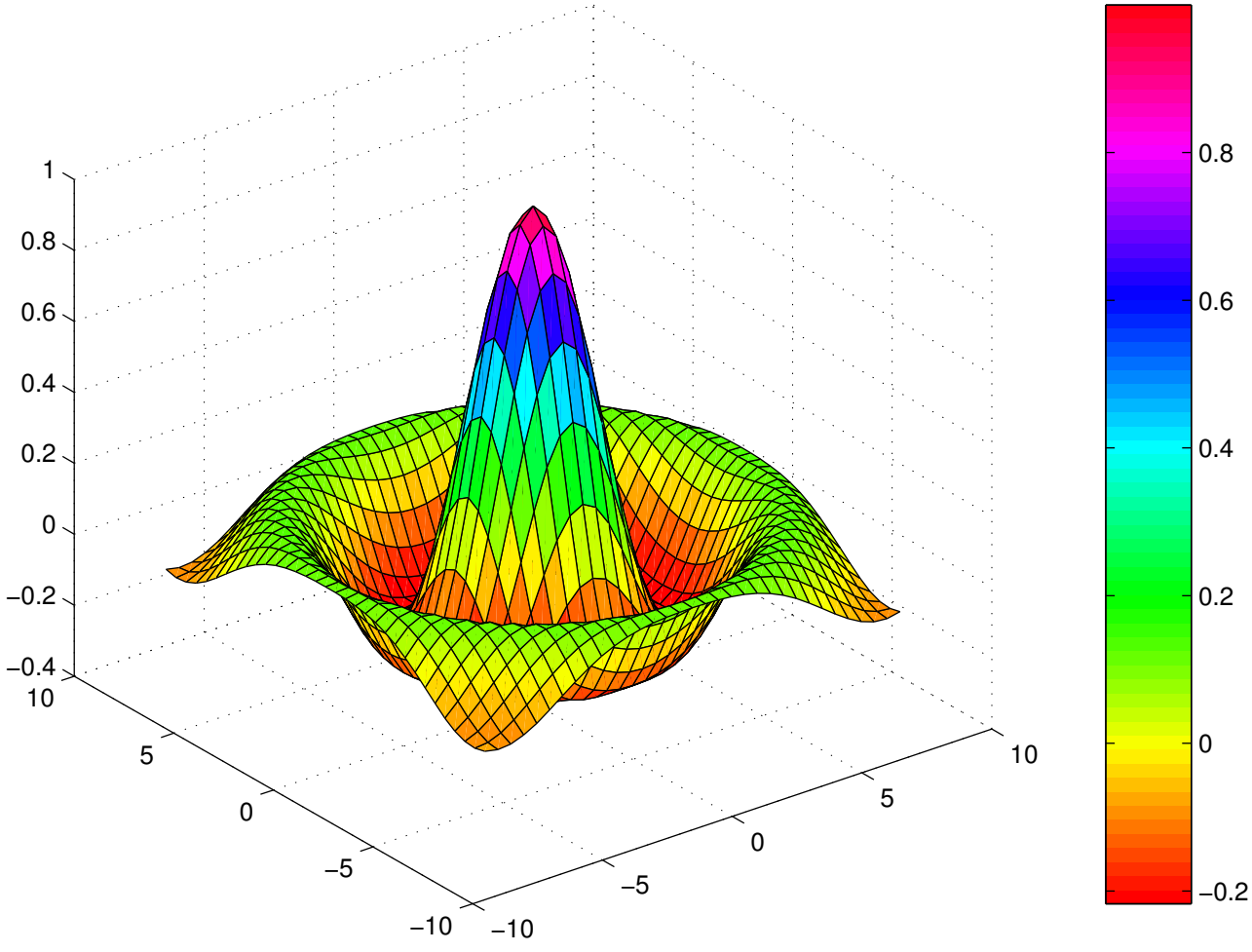


Surface Plots

Let us reuse the last example, but make a colored surface plot instead

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
surf(X,Y,Z)
colormap hsv
colorbar
```

Surface Plot



Programming Advice

Strive for order and logic in your program. Use logical operators, tests, loops and functions where appropriate. For example, the statement:

$$if \quad a = 0 \quad or \quad a > 2, \quad then \quad b = 1 \tag{6}$$

is written in Matlab code as

```
if a == 0 | a > 2
   b = 1;
end
```

Loops

A **while** loop runs until some condition is met. In this example the loop run as long as $a \in (0.1, 0.9)$

```
a = 0.5;
while ((a > 0.1) & (a < 0.9))
   a = rand;
   n = n+1;
end
```

Another type of loop is the **for** loop, which runs a given number of rounds.

```
for n = 1:N
   a(n) = 1/n;
end
```

Functions

Sometimes there are no built-in functions for what you need. Then you can create your own functions.
Say we want to define this function in matlab:

$$f(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6 \qquad (7)$$

That can be done like this:

```
function y = humps(x)
     y = 1./((x - 0.3).^2 + 0.01) +
         1./((x - 0.9).^2 + 0.04) - 6;
```

- A function is an m-file beginning with the word 'function'
- A function has a user-specified input and output
- A function has its own 'local' workspace. Variables defined in the function are not stored in the ordinary workspace. Nor are your workspace variables altered if given a new value in the 'local' workspace. **A function does not assign any values to anything**
- Exception to this rule is if you define a **global** variable. Type **'help global'** for more information
- You can have several functions stored in one file. The 'subfunctions' work exactly as a normal function, but can not be axcessed directly from the command window (unlike other languages).

General Advice

- Use built-in functions. Safer and more efficient
- Use built-in documentation. See the help menu.
- Try to avoid nested loops. It is inefficient.
- Document your code. Add comments.
- Logical structure

Example: Diffusion equation

The diffusion equation (also known as the heat equation) have this form:

$$u_t = u_{xx} + u_{yy} \tag{8}$$

where $u_t = \partial u/\partial t$, $u_{xx} = \partial^2 u/\partial x^2$, and so on.

The equation can model a temperature u at a given time t at position (x, y) in a 2D plane.

Given suitable boundary conditions and initial condition we can find a numerical solution for this equation.

An algorithm for this equation is:

```
1. Define geometry
2. Set constants and variables
3. Set initial condition
4. loop until t >= end_time:
   4.1. Calculate inner points
   4.2. Set boundary conditions
   4.3. plot u(x,t) at this level
   4.4. t = t + dt
```