

MATLAB Introduction Course: Lecture 3

Øivind K. Kjerstad

10. October 2014

TOC

- 1 Linear Algebra
- 2 Polynomials
- 3 Optimization
- 4 Differentiation/Integration
- 5 Differential Equations

Solving Linear Equations

- Given a system of linear equations

$$x + 2y - 3z = 5$$

$$-3x - y + z = -8$$

$$x - y + z = 0$$

- Construct matrices so the system is described by $Ax = b$
 - `>> A=[1 2 -3;-3 -1 1;1 -1 1]`
 - `>> b=[5; -8; 0]`
- Solve with a single line of code!
 - `>> x = A\b`
 - `x` is a 3x1 vector containing the values of x , y , and z
- The `\` will work with square or rectangular systems
- Gives least squares solution for rectangular systems. Solution depends on whether the system is over or underdetermined

Solving Linear Equations

- Given a matrix
 - ▶ `>> mat=[1 2 -3;-3 -1 1;1 -1 1]`
- The rank of a matrix
 - ▶ `rank(mat)`
 - ▶ The number of linearly independent rows or columns
- The determinant
 - ▶ `det(mat)`
 - ▶ `mat` must be square
 - ▶ If the determinant is nonzero, then the matrix is invertible
- The matrix inverse
 - ▶ `inv(mat)`
 - ▶ if an equation is of the form $Ax = b$ with A a square matrix, $x = A \setminus b$ is the same as `x=inv(A)*b`

Matrix Decompositions

- MATLAB has several built-in matrix decomposition methods
- The most common are:
 - ▶ $[V,D]=\text{eig}(X)$
 - ★ Eigenvalue decomposition
 - ▶ $[U,S,V]=\text{svd}(X)$
 - ★ Singular value decomposition
 - ▶ $[Q,R]=\text{qr}(X)$
 - ★ QR decomposition
- For more information, see [help\doc](#)

Exercise 1: Solve Linear Systems

Exercise 1a

The linear system:

$$x + 4y = 34$$

$$-3x + y = 2$$

- Determine the rank of the problem
- Solve for x and y

Exercise 1b

The linear system:

$$2x - 2y = 4$$

$$-x + y = 3$$

$$3x + 4y = 2$$

- Determine the rank of the problem
- Solve for x and y
- Determine the least squares error

Solution

Exercise 1a

```
A=[1 4;-3 1];  
b=[34;2];  
rank(A)  
x=inv(A)*b;
```

Exercise 1b

```
A=[2 -2;-1 1;3 4];  
b=[4;3;2];  
rank(A)  
x=A\b  
error=abs(A*x-b)
```

TOC

- 1 Linear Algebra
- 2 Polynomials**
- 3 Optimization
- 4 Differentiation/Integration
- 5 Differential Equations

Polynomials

- Many functions can be well described by a high-order polynomial
- MATLAB represents a polynomials by a vector of coefficients
 - ▶ $p(x) = ax^3 + bx^2 + cx + d$
 - ▶ $P = [a \ b \ c \ d]$

Examples

$$p(x) = x^2 - 2 \Rightarrow P = [1 \ 0 \ -2]$$

$$p(x) = 2x^3 \Rightarrow P = [2 \ 0 \ 0 \ 0]$$

Polynomial Operations

- P is a vector of length $N+1$ describing an N -th order polynomial
- Polynomial roots
 - ▶ `>> r = roots(P)`
 - ▶ `r` is a vector of length N
- Polynomial from the roots
 - ▶ `>> P = poly(r)`
 - ▶ `r` is a vector of length N
- Evaluate a polynomial at a point
 - ▶ `>> y0 = polyval(P,x0)`
 - ▶ `x0` is a single value; `y0` is a single value
- Evaluate a polynomial at many points
 - ▶ `>> y = polyval(P,x)`
 - ▶ `x` is a vector; `y` is a vector

Polynomial Fitting

- MATLAB makes it very easy to fit polynomials to data
 - ▶ `polyfit`

Example

Given data vectors $X = [-1 \ 0 \ 2]$ and $Y = [0 \ -1 \ 3]$

```
p = polyfit(X,Y,2)
plot(X,Y,'o', 'MarkerSize', 10);
hold on;
plot(-3:.01:3,polyval(p,-3:.01:3), 'r--');
```

This finds the best second order polynomial that fits the points $(-1,0)$, $(0,-1)$, and $(2,3)$. See [doc polyfit](#) for more info

MATLAB has a toolbox for fitting data to expressions, see [cftool](#) and [splinetool](#)

Exercise 2: Polynomial Fitting

Polynomial Fitting

- Evaluate $y = x^2$ for `x=-4:0.1:4`
- Add random noise to `y`, use `randn`
- Fit a 2nd degree polynomial to the noisy data
- Plot the noisy data using circular markers and the fitted polynomial using a solid red line

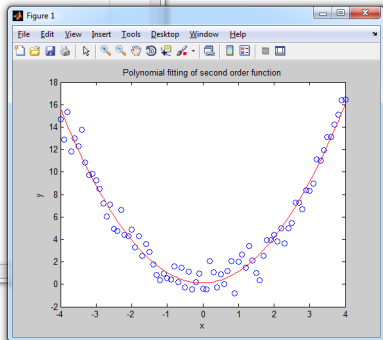
Solution

```
C:\Users\oivindka\Documents\MATLAB\Matlab_Course\Lecture_3\Ex_3.m

EDITOR PUBLISH VIEW
New Open Save Find Files Compare Comment Insert % Find Files %
Print Indent Go To Breakpoints Run Run and Advance Run and Time
FILE EDIT NAVIGATE BREAKPOINTS RUN

1 % Evaluate y = x^2
2 x=-4:0.1:4;
3 y=x.^2;
4
5 % Adding noise to the data
6 y=y+randn(size(y));
7
8 % Fit polynomial to data
9 p=polyfit(x,y,2);
10
11 % Plotting
12 plot(x,y,'o');
13 hold on;
14 plot(x,polyval(p,x),'r')
15 xlabel('x')
16 ylabel('y')
17 title('Polynomial fitting of second order function')

script
```

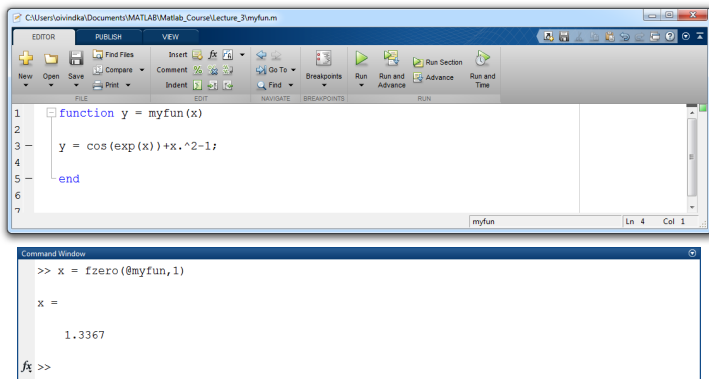


TOC

- 1 Linear Algebra
- 2 Polynomials
- 3 Optimization**
- 4 Differentiation/Integration
- 5 Differential Equations

Nonlinear Root Finding

- Many real-world problems require us to solve $f(x) = 0$
- Can use `fzero` to calculate roots for any arbitrary function
- `fzero` needs a function passed to it
 - ▶ We will see this type of operation more as we go into solving equations
 - ▶ `>> x=fzero(@myfun,x0)`



The screenshot shows the MATLAB Editor window with a file named 'myfun.m' open. The code in the editor is as follows:

```
1 function y = myfun(x)
2
3     y = cos(exp(x))+x.^2-1;
4
5 end
```

Below the editor is the Command Window, which shows the execution of the `fzero` function:

```
>> x = fzero(@myfun,1)

x =

    1.3367

fx >>
```

Minimizing a Function

- `fminbnd`: minimizing a function over a bounded interval
 - ▶ `>> x=fminbnd(@myfun,-1,2);`
 - ★ `myfun` takes a scalar input and returns a scalar output
 - ★ `myfun` find the minimum in the interval $-1 \leq x \leq 2$
- `fminsearch`: unconstrained interval
 - ▶ `>> x=fminsearch(@myfun,.5);`
 - ★ finds the local minimum of `myfun` starting at $x = 0.5$

Anonymous Functions

- What if `myfun` is relatively simple?
 - ▶ Then, using anonymous functions can be more efficient and simpler
 - ▶ In practice this is writing the function directly into the function call
- `>> x=fzero(@(input)(function expression),x0)`

Examples

```
>> x=fzero(@(x)(cos(exp(x))+x^2-1),x0)
>> x=fminbnd(@(x)(cos(exp(x))+x^2-1),x_low,x_high);
>> x=fminsearch(@(x)(cos(exp(x))+x^2-1),x0);
```

Optimization Toolbox

- If you are familiar with optimization methods, use the optimization toolbox
 - ▶ Useful for larger, more structured optimization problems
 - ▶ It is located under MATLAB apps
- Sample functions (see [help](#) for more info)
 - ▶ `linprog`
 - ★ Linear programming using interior point methods
 - ▶ `quadprog`
 - ★ Quadratic programming solver
 - ▶ `fmincon`
 - ★ Constrained nonlinear optimization

Optimization Toolbox

The screenshot displays the MATLAB Optimization Tool interface. The main window is titled "Optimization Tool" and contains several panels:

- Problem Setup and Results:** Shows the solver set to "Mincon - Constrained nonlinear minimization" and the algorithm set to "Interior-point". The objective function is "Approximated by solver".
- Options:** A large section for configuring solver parameters. Key options include:
 - Stopping criteria:** Max iterations (1000), Max function evaluations (3000), X tolerance (1e-10), Function tolerance (1e-6), Constraint tolerance (1e-6), SQP constraint tolerance (1e-6), and Unboundedness threshold (-1e20).
 - Function value check:** Error if user-supplied function returns Inf, NaN or complex.
 - User-supplied derivatives:** Validate user-supplied derivatives.
 - Hessian sparsity pattern:** Use default sparse(ones(numberOfVariables)).
 - Hessian multiply function:** Use default: No multiply function.
 - Approximated derivatives:** Finite differences (f = F(x) - F(x-h)). Type: forward differences. Relative perturbation vector ϵ (1e-10), Minimum perturbation $(\delta^*)_i$ (0), and Maximum perturbation $(\delta^*)_i$ (Inf).
- Quick Reference:** A sidebar on the right providing a summary of the Mincon Solver, including its purpose (finding a minimum of a constrained nonlinear multivariable function) and a list of sections: Problem Setup and Results, Options, Suggested Next Steps, and More Information.

Exercise 3: Find Minimum

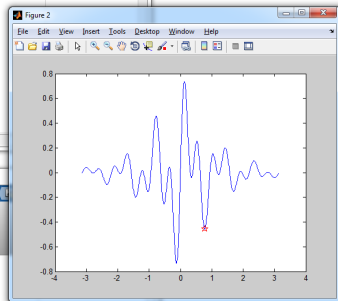
Exercise 3: Find Minimum

- Find the minimum of the function $f(x) = \cos(x)\sin(10x)e^{|x|}$ over the range $x \in [-\pi \pi]$ using `fminbnd`
- Plot the function for the given range
- Plot the found minimum solution in the same figure

Solution

```
C:\Users\loivindka\Documents\MATLAB\Matlab_Course\Lecture_3\myFunc5.m
EDITOR PUBLISH VIEW
+ Find Files Insert
New Open Save Compare Comment
Print Indent Go To Breakpoints Run Run and Advance Run Section Run and Time
FILE EDIT NAVIGATE BREAKPOINTS RUN
1
2 x0 = fminbnd(@myFunc, -pi, pi);
3
4 x=-pi:.01:pi;
5
6 figure;
7 plot(x, myFunc(x));
8 hold on;
9 plot(x0, myFunc(x0), 'rp', 'MarkerSize', 10);
script
```

```
C:\Users\loivindka\Documents\MATLAB\Matlab_Course\Lecture_3\myfun.m
EDITOR PUBLISH VIEW
+ Find Files Insert
New Open Save Compare Comment
Print Indent Go To Breakpoints Run Run and Advance Run Section Run and Time
FILE EDIT NAVIGATE BREAKPOINTS RUN
1 function y = myfun(x)
2
3 y = cos(exp(x)) + x.^2 - 1;
4
5 end
6
myfun Ln 5 Col 4
```



Remember to check what built-in functions do!

TOC

- 1 Linear Algebra
- 2 Polynomials
- 3 Optimization
- 4 Differentiation/Integration**
- 5 Differential Equations

Numerical Differentiation

- MATLAB can differentiate numerically using `diff`
 - ▶ `diff` computes the first difference
- `diff` also works on matrices
 - ▶ Computes the first difference along the 2nd dimension
 - ▶ The opposite of `diff` is the cumulative sum `cumsum`
 - ▶ See help for more details
- For the 2D gradient, see `gradient`

Example

```
x=0:0.01:2*pi;  
y=sin(x);  
dydx=diff(y)./diff(x);
```

Numerical Integration

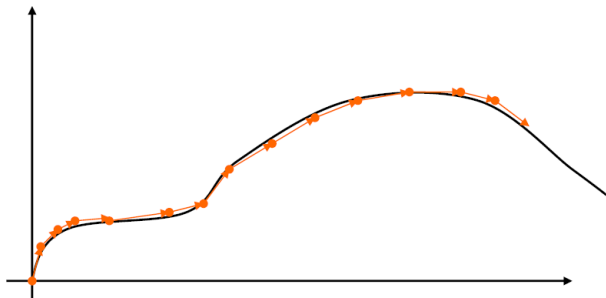
- MATLAB includes the most common integration methods
- Adaptive Simpson's quadrature (input is a function)
 - ▶ `q=quad('myFun',0,10);`
 - ▶ q is the integral of the function myFun from 0 to 10
 - ▶ `q2=quad(@(x) sin(x)*x,0,pi);`
 - ▶ q2 is the integral of $\sin(x)x$ from 0 to π
- Trapezoidal rule (input is a vector)
 - ▶ `>> x=0:0.01:pi;`
 - ▶ `>> z=trapz(x,sin(x));`
 - ▶ z is the integral of $\sin(x)$ from 0 to π
 - ▶ `>>z2=trapz(x,sqrt(exp(x))./x);`
 - ▶ z2 is the integral of $\sqrt{\frac{e^x}{x}}$ from 0 to π

TOC

- 1 Linear Algebra
- 2 Polynomials
- 3 Optimization
- 4 Differentiation/Integration
- 5 Differential Equations**

ODE Solvers: Method Overview

- Given a differential equation, the solution can be found by integration:



- ▶ Evaluate the derivative at a point and approximate by straight line
- ▶ Errors accumulate!
- ▶ Variable timestep can decrease the number of iterations

ODE Solvers: MATLAB

- MATLAB contains implementations of common ODE solvers
- Using the correct ODE solver can save time and give more accurate results
 - ▶ `ode23`
 - ★ Low-order solver. Use when integrating over small intervals or when accuracy is less important than speed
 - ▶ `ode45`
 - ★ High order (Runge-Kutta) solver. High accuracy and reasonable speed. Most commonly used
 - ▶ `ode15s`
 - ★ Stiff ODE solver (Gear's algorithm), use when the diff eq's have time constants that vary by orders of magnitude

ODE Solvers: Standard Syntax

The ODE function call

```
[t,y]=ode45('myODE',[0,10],x0)
```

- ode45 is the solver
 - 'myODE' is the function to be evaluated
 - [0, 10] is the simulation time range
 - x0 is the initial conditions
-
- Inputs
 - ▶ ODE function name (or anonymous function). This function takes inputs (t,y), and returns dy/dt
 - ▶ Time interval: 2-element vector specifying initial and final time
 - ▶ Initial conditions: column vector with an initial condition for each ODE. This is the first input to the ODE function
 - Outputs
 - ▶ t contains the time points
 - ▶ y contains the corresponding values of the integrated variables.

ODE Example: The pendulum

Example: The pendulum equations

$$\ddot{\theta} + \frac{g}{L} \sin(\theta) = 0$$

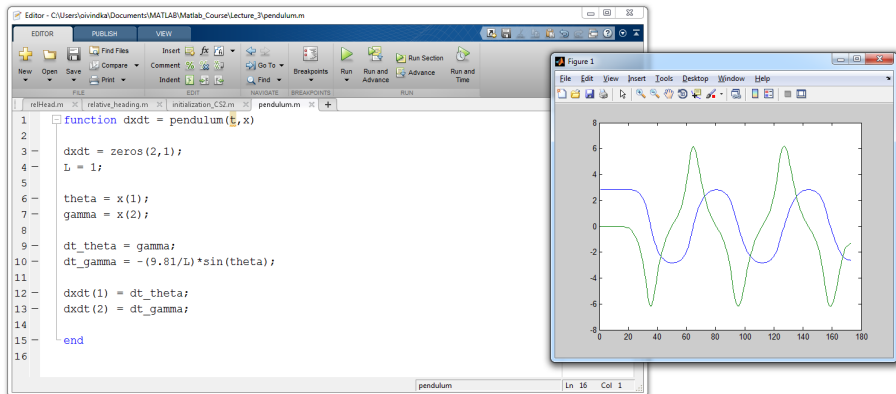
Define the state $x = [\theta \ \gamma]^T$, where $\dot{\theta} = \gamma$. Then we can write the system as

$$\dot{\theta} = \gamma$$

$$\dot{\gamma} = -\frac{g}{L} \sin(\theta)$$

- Must make into a system of first-order equations to use ODE solvers
- Nonlinear equations are OK!

ODE Example: The pendulum



```
Command Window
>> [t,x]=ode45('pendulum',[0 10],[0.9*pi 0]);
>> plot(x)
fx >>
```

ODE Solvers: Custom Options

- MATLAB's ODE solvers use a variable timestep
- Sometimes a fixed timestep is desirable
 - ▶ `[t,y]=ode45('myODE',[0:0.001:0.5],x0);`
 - ★ Specify the timestep by giving a vector of times
 - ★ The function value will be returned at the specified points
 - ★ Fixed timestep is usually slower because function values are interpolated to give values at the desired timepoints
- You can customize the error tolerances using `odeset`
 - ▶ `options=odeset('RelTol',1e-6,'AbsTol',1e-10);`
 - ▶ `[t,y]=ode45('myODE',[0 100],x0,options);`
 - ★ This guarantees that the error at each step is less than RelTol times the value at that step, and less than AbsTol
 - ★ Decreasing error tolerance can considerably slow the solver
 - ★ See doc odeset for a list of options you can customize

Exercise 4: ODE

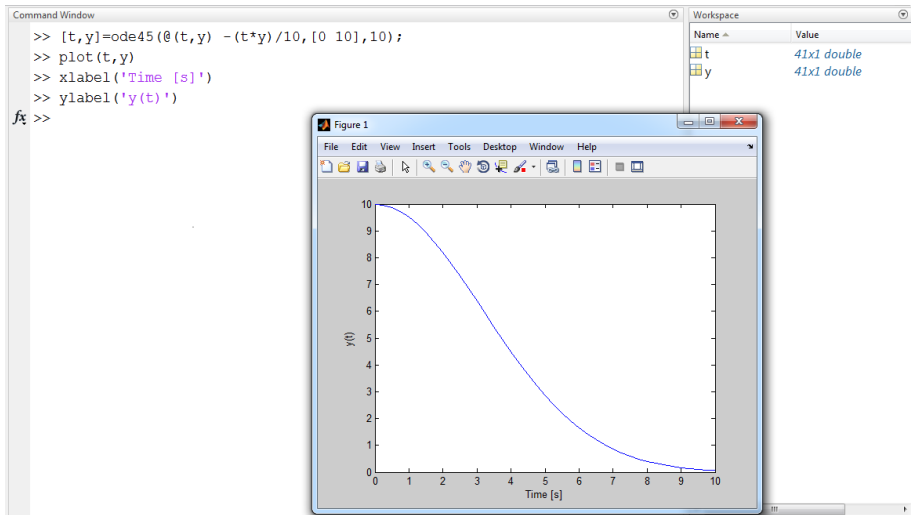
Exercise 4: ODE

- Implement the ODE:

$$\frac{dy}{dt} = -\frac{ty}{10}$$

- with initial condition $y(0) = 10$ over the interval $t = [0, 10]$
- Use `ode45`
- Plot $y(t)$

Exercise 4: Solution



>> THE END