

Outline

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

Systems of Linear Equations

MATLAB makes linear algebra fun!

- 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];`
- And solve with a single line of code!
 - » `x=A\b;`
 - x is a 3×1 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.

Worked Example: Linear Algebra

- Solve the following systems of equations:

➤ System 1:

$$x + 4y = 34$$

$$-3x + y = 2$$

» `A=[1 4;-3 1];`

» `b=[34;2];`

» `rank(A)`

» `x=inv(A)*b;`

» `x=A\b;`

➤ System 2:

$$2x - 2y = 4$$

$$-x + y = 3$$

$$3x + 4y = 2$$

» `A=[2 -2;-1 1;3 4];`

» `b=[4;3;2];`

» `rank(A)`

➤ rectangular matrix

» `x=A\b;`

➤ gives least squares solution

» `error=abs(A*x1-b)`

More Linear Algebra

- Given a matrix
 - » `mat=[1 2 -3;-3 -1 1;1 -1 1];`
- Calculate the rank of a matrix
 - » `r=rank(mat);`
 - the number of linearly independent rows or columns
- Calculate the determinant
 - » `d=det(mat);`
 - mat must be square; matrix invertible if det nonzero
- Get the matrix inverse
 - » `E=inv(mat);`
 - if an equation is of the form $A*x=b$ with A a square matrix, $x=A\backslash b$ is (mostly) the same as $x=inv(A)*b$
- Get the condition number
 - » `c=cond(mat);` (or its reciprocal: `c = rcond(mat);`)
 - if condition number is large, when solving $A*x=b$, small errors in b can lead to large errors in x (optimal $c=1$)

Matrix Decompositions

- MATLAB has many built-in matrix decomposition methods
- The most common ones are
 - » `[V,D]=eig(X)`
 - Eigenvalue decomposition
 - » `[U,S,V]=svd(X)`
 - Singular value decomposition
 - » `[Q,R]=qr(X)`
 - QR decomposition
 - » `[L,U]=lu(X)`
 - LU decomposition
 - » `R=chol(X)`
 - Cholesky decomposition (R must be positive definite)

Exercise: Fitting Polynomials

- Find the best second-order polynomial that fits the points: $(-1,0)$, $(0,-1)$, $(2,3)$.

$$a(-1)^2 + b(-1) + c = 0$$

$$a(0)^2 + b(0) + c = -1$$

$$a(2)^2 + b(2) + c = 3$$

Outline

- (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
 - if vector P describes a polynomial

$$ax^3+bx^2+cx+d$$

$P(1)$ $P(2)$ $P(3)$ $P(4)$

- $P=[1 \ 0 \ -2]$ represents the polynomial x^2-2
- $P=[2 \ 0 \ 0 \ 0]$ represents the polynomial $2x^3$

Polynomial Operations

- P is a vector of length N+1 describing an N-th order polynomial
- To get the roots of a polynomial
 - » `r=roots(P)`
 - r is a vector of length N
- Can also get the polynomial from the roots
 - » `P=poly(r)`
 - r is a vector length N
- To evaluate a polynomial at a point
 - » `y0=polyval(P,x0)`
 - x0 is a single value; y0 is a single value
- To evaluate a polynomial at many points
 - » `y=polyval(P,x)`
 - x is a vector; y is a vector of the same size

Polynomial Fitting

- MATLAB makes it very easy to fit polynomials to data
- Given data vectors $X = [-1 \ 0 \ 2]$ and $Y = [0 \ -1 \ 3]$
 - » `p2=polyfit(X,Y,2);`
 - finds the best (least-squares sense) second-order polynomial that fits the points $(-1,0)$, $(0,-1)$, and $(2,3)$
 - see **help polyfit** for more information
 - » `plot(X,Y,'o', 'MarkerSize', 10);`
 - » `hold on;`
 - » `x = -3:.01:3;`
 - » `plot(x,polyval(p2,x), 'r--');`

Exercise: Polynomial Fitting

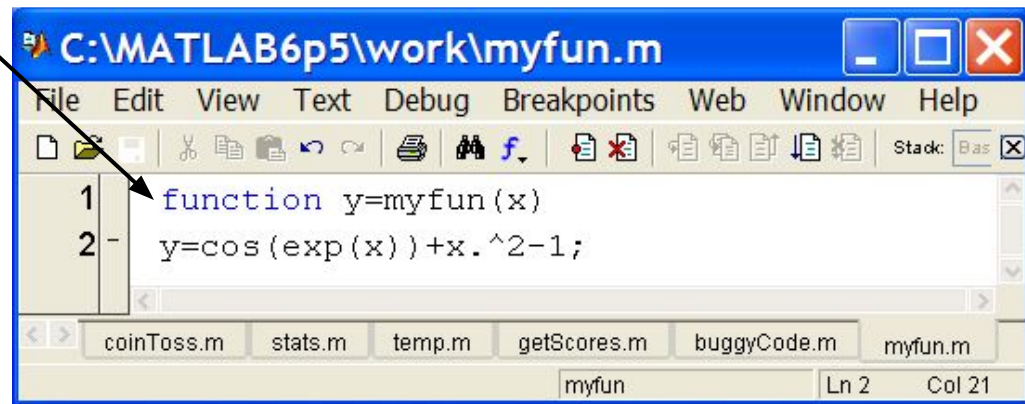
- Evaluate $y = x^2$ for $x = -4:0.1:4$.
- Add random noise to these samples. Use **randn**. Plot the noisy signal with `.` markers
- Fit a 2nd degree polynomial to the noisy data
- Plot the fitted polynomial on the same plot, using the same x values and a red line

Outline

- (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 more and more as we delve into solving equations.
- Make a separate function file
 - » `x=fzero('myfun',1)`
 - » `x=fzero(@myfun,1)`
 - 1 specifies a point close to where you think the root is



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(x) will be the minimum of myfun for $-1 \leq x \leq 2$
- **fminsearch**: unconstrained interval
 - » `x=fminsearch('myfun',.5)`
 - finds the local minimum of myfun starting at $x=0.5$
- Maximize $g(x)$ by minimizing $f(x)=-g(x)$
- Solutions may be local!

Anonymous Functions

- You do not have to make a separate function file

- » `x=fzero(@myfun,1)`

- What if myfun is really simple?

- Instead, you can make an anonymous function

- » `x=fzero(@ (x) (cos(exp(x))+x.^2-1), 1);`

input function to evaluate

- » `x=fminbnd(@ (x) (cos(exp(x))+x.^2-1), -1, 2);`

- Can also store the function handle

- » `func=@ (x) (cos(exp(x))+x.^2-1);`

- » `func(1:10);`

Optimization Toolbox

- If you are familiar with optimization methods, use the optimization toolbox
- Useful for larger, more structured optimization problems
- Sample functions (see [help](#) for more info)
 - » [linprog](#)
 - linear programming using interior point methods
 - » [quadprog](#)
 - quadratic programming solver
 - » [fmincon](#)
 - constrained nonlinear optimization

Exercise: Min-Finding

- Find the minimum of the function $f(x) = \cos(4x)\sin(10x)e^{-|x|}$ over the range $-\pi$ to π . Use `fminbnd`.
- Plot the function on this range to check that this is the minimum.

Digression: Numerical Issues

- Many techniques in this lecture use floating point numbers
- **This is an approximation!**
- Examples:
 - » `sin(pi) = ?`
 - » `sin(2 * pi) = ?`
 - » `sin(10e16 * pi) = ?`
 - Both sin and pi are approximations!
 - » `A = (10e13)*ones(10) + rand(10)`
 - A is nearly singular, poorly conditioned (see `cond(A)`)
 - » `inv(A)*A = ?`

A Word of Caution

- MATLAB knows no fear!
- Give it a function, it optimizes / differentiates / integrates
 - That's great! It's so powerful!
- Numerical techniques are powerful **but** not magic
- **Beware of overtrusting the solution!**
 - You will get an answer, but it may not be what you want
- Analytical forms may give more intuition
 - Symbolic Math Toolbox

Outline

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

Numerical Differentiation

- MATLAB can 'differentiate' numerically

- » `x=0:0.01:2*pi;`

- » `y=sin(x);`

- » `dydx=diff(y)./diff(x);`

- `diff` computes the first difference

- Can also operate on matrices

- » `mat=[1 3 5;4 8 6];`

- » `dm=diff(mat,1,2)`

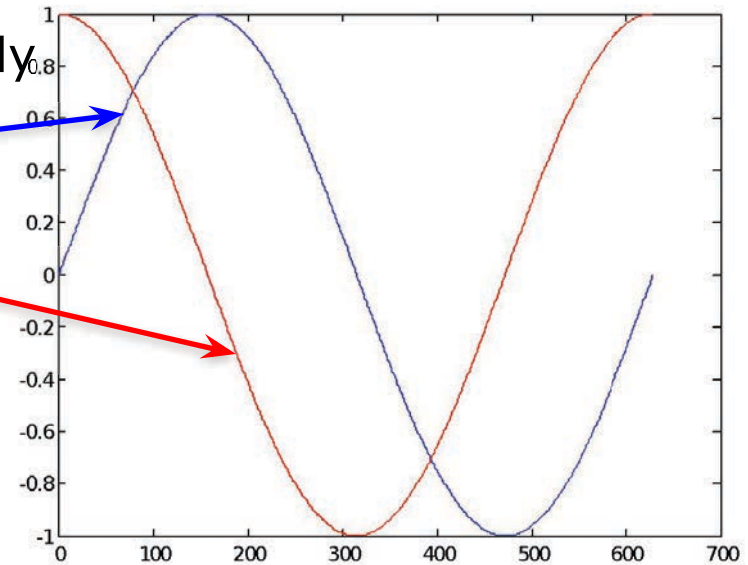
- first difference of `mat` along the 2nd dimension, `dm=[2 2;4 -2]`

- The opposite of `diff` is the cumulative sum `cumsum`

- 2D gradient

- » `[dx,dy]=gradient(mat);`

- Higher derivatives / complicated problems: Fit spline (see **help**)



Numerical Integration

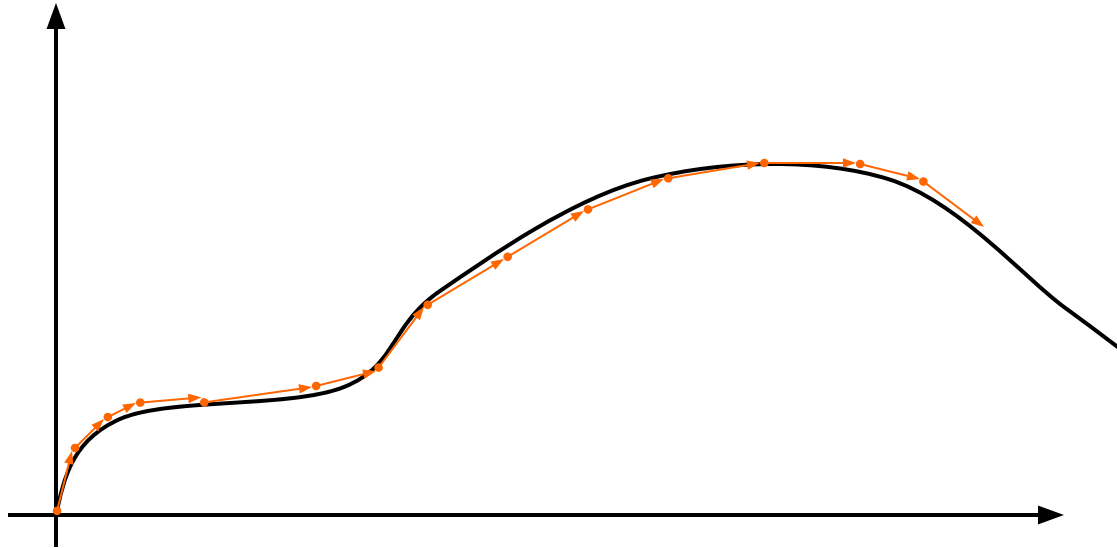
- MATLAB contains common integration methods
- Adaptive Simpson's quadrature (input is a function)
 - » `q=quad('myFun',0,10)`
 - q is the integral of the function `myFun` from 0 to 10
 - » `q2=quad(@(x) sin(x).*x,0,pi)`
 - q2 is the integral of `sin(x).*x` from 0 to pi
- Trapezoidal rule (input is a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

Outline

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

ODE Solvers: Method

- 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 you lots of 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

- To use standard options and variable time step

» `[t,y]=ode45('myODE',[0,10],[1;0])`

ODE integrator:
23, 45, 15s

ODE function

Time range

Initial conditions

- Inputs:
 - ODE function name (or anonymous function). This function should take inputs (t,y), and returns dy/dt
 - Time interval: 2-element vector with initial and final time
 - Initial conditions: column vector with an initial condition for each ODE. This is the first input to the ODE function
 - Make sure all inputs are in the same (variable) order
- Outputs:
 - t contains the time points
 - y contains the corresponding values of the variables

ODE Function

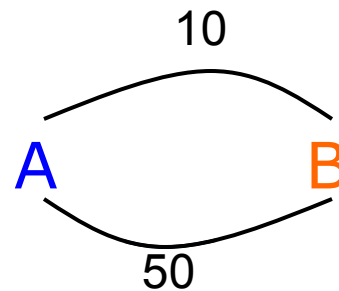
- The ODE function must return the value of the derivative at a given time and function value

- Example: chemical reaction

➤ Two equations

$$\frac{dA}{dt} = -10A + 50B$$

$$\frac{dB}{dt} = 10A - 50B$$



➤ ODE file:

- y has [A;B]
- dydt has [dA/dt;dB/dt]

```
C:\MATLAB6p5\work\chem.m
File Edit View Text Debug Breakpoints Web Window Help
1 % chem: chemical reaction ode function
2 function dydt=chem(t,y)
3     dydt=zeros(2,1);
4     dydt(1)=-10*y(1)+50*y(2);
5     dydt(2)=10*y(1)-50*y(2);
```

ODE Function: viewing results

- To solve and plot the ODEs on the previous slide:

- » `[t,y]=ode45('chem',[0 0.5],[0 1]);`

- » assumes that only chemical B exists initially

- » `plot(t,y(:,1),'k','LineWidth',1.5);`

- » `hold on;`

- » `plot(t,y(:,2),'r','LineWidth',1.5);`

- » `legend('A','B');`

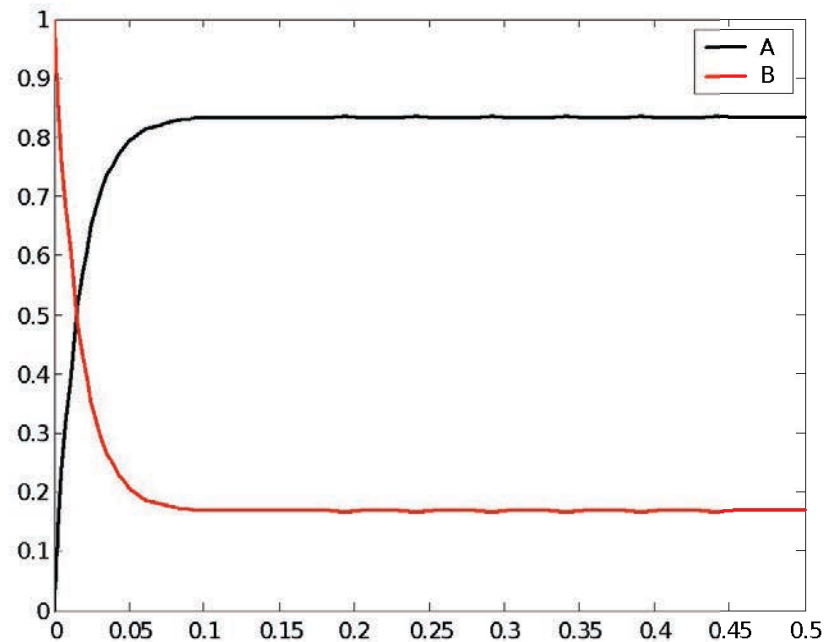
- » `xlabel('Time (s)');`

- » `ylabel('Amount of chemical (g)');`

- » `title('Chem reaction');`

ODE Function: viewing results

- The code on the previous slide produces this figure



Higher Order Equations

- Must make into a system of first-order equations to use ODE solvers
- Nonlinear is OK!
- Pendulum example:

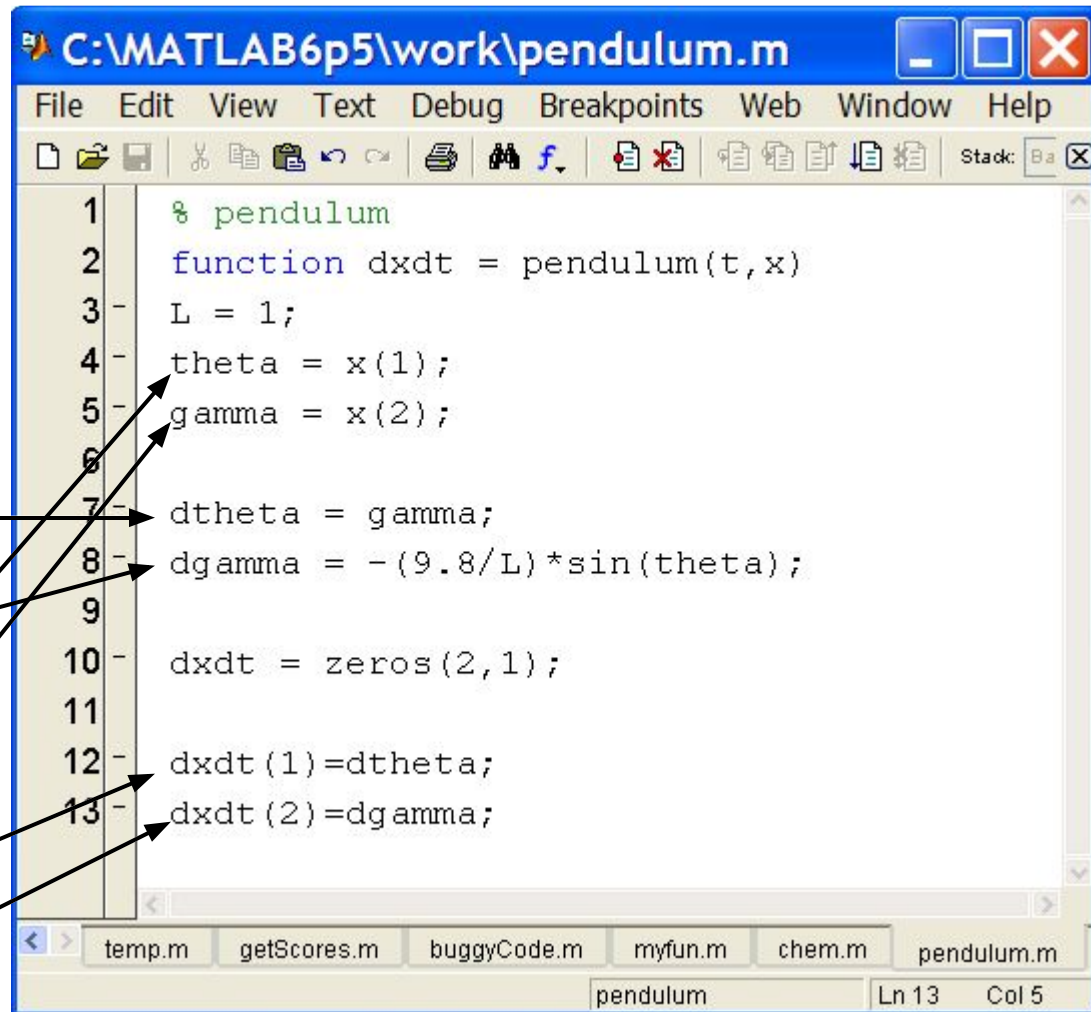
$$\ddot{\theta} + \frac{g}{L} \sin(\theta) = 0$$

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

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

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

$$\bar{x} = \begin{bmatrix} \theta \\ \gamma \end{bmatrix}$$
$$\frac{d\bar{x}}{dt} = \begin{bmatrix} \dot{\theta} \\ \dot{\gamma} \end{bmatrix}$$

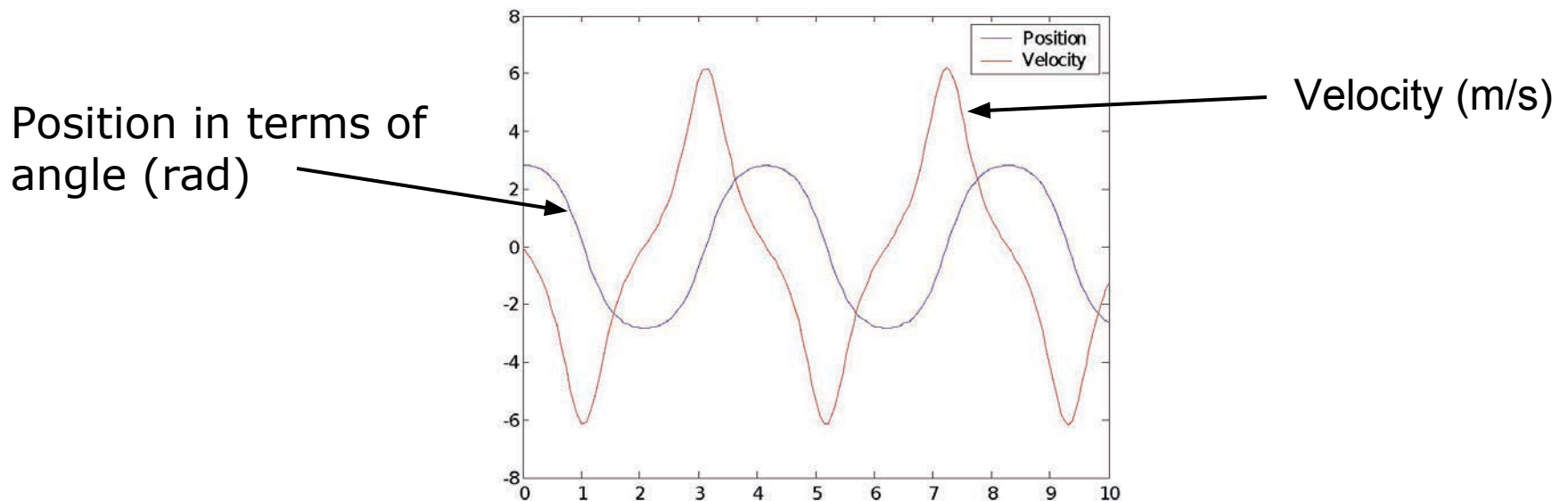


A screenshot of a MATLAB script editor window titled 'C:\MATLAB6p5\work\pendulum.m'. The script defines a function 'pendulum' that takes time 't' and state 'x' as inputs and returns the derivative 'dxdt'. The state 'x' is a 2x1 vector where 'x(1)' is the angle 'theta' and 'x(2)' is the angular velocity 'gamma'. The script sets 'L = 1' and calculates the derivative components: 'dtheta = gamma' and 'dgamma = -(9.8/L)*sin(theta)'. These are then assigned to 'dxdt(1)' and 'dxdt(2)' respectively. The script is shown with line numbers 1 through 13. Arrows from the mathematical equations on the left point to the corresponding lines in the script: from the definition of 'gamma' to line 7, from the second derivative equation to line 8, and from the state vector definition to lines 4 and 5.

```
1 % pendulum
2 function dxdt = pendulum(t,x)
3 L = 1;
4 theta = x(1);
5 gamma = x(2);
6
7 dtheta = gamma;
8 dgamma = -(9.8/L)*sin(theta);
9
10 dxdt = zeros(2,1);
11
12 dxdt(1)=dtheta;
13 dxdt(2)=dgamma;
```

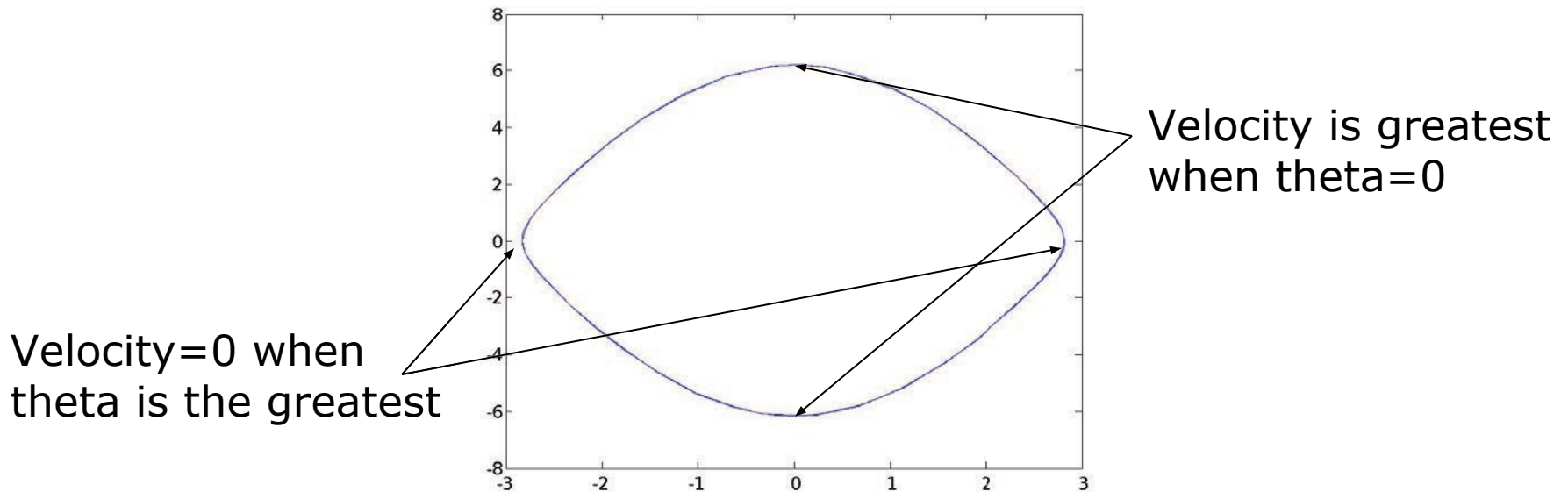
Plotting the Output

- We can solve for the position and velocity of the pendulum:
 - » `[t,x]=ode45('pendulum',[0 10],[0.9*pi 0]);`
 - assume pendulum is almost horizontal
 - » `plot(t,x(:,1));`
 - » `hold on;`
 - » `plot(t,x(:,2),'r');`
 - » `legend('Position','Velocity');`



Plotting the Output

- Or we can plot in the phase plane:
 - » `plot(x(:,1),x(:,2));`
 - » `xlabel('Position');`
 - » `yLabel('Velocity');`
- The phase plane is just a plot of one variable versus the other:



ODE Solvers: Custom Options

- MATLAB's ODE solvers use a variable timestep
- Sometimes a fixed timestep is desirable
 - » `[t,y]=ode45('chem',[0:0.001:0.5],[0 1]);`
 - Specify timestep by giving a vector of (increasing) times
 - The function value will be returned at the specified points
- You can customize the error tolerances using `odeset`
 - » `options=odeset('RelTol',1e-6,'AbsTol',1e-10);`
 - » `[t,y]=ode45('chem',[0 0.5],[0 1],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: ODE

- Use `ode45` to solve for $y(t)$ on the range $t=[0 \ 10]$, with initial condition $y(0)=10$ and $dy/dt = -t y/10$
- Plot the result.

Exercise: ODE

- Use `ode45` to solve for $y(t)$ on the range $t=[0 \ 10]$, with initial condition $y(0)=10$ and $dy/dt = -t y/10$
- Plot the result.
- Make the following function
 - » `function dydt=odefun(t,y)`
 - » `dydt=-t*y/10;`
- Integrate the ODE function and plot the result
 - » `[t,y]=ode45('odefun',[0 10],10);`
- Alternatively, use an anonymous function
 - » `[t,y]=ode45(@(t,y) -t*y/10,[0 10],10);`
- Plot the result
 - » `plot(t,y);xlabel('Time');ylabel('y(t)');`

Exercise: ODE

- The integrated function looks like this:

