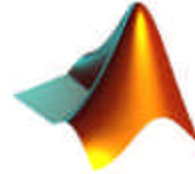


Matlab, Numerical Integration, and Simulation

- **Matlab tutorial**
 - Basic programming skills
 - Visualization
 - Ways to look for help
- **Numerical integration**
 - Integration methods: explicit, implicit; one-step, multi-step
 - Accuracy and numerical stability
 - Stiff systems
- **Programming examples**
 - Solutions to HW0 using Matlab
 - Mass-spring-damper system



MATLAB

- **MATLAB (MATrix LABoratory) is an interpretative (interactive) programming language**
 - control flow statements, functions, data structures, input/output, and object-oriented programming features
- **MATLAB working environment.**
 - tools for developing, managing, debugging, and profiling M-files
- **MATLAB graphics system**
 - 2D and 3D data visualization, image processing, animation, and presentation graphics
- **MATLAB mathematical function library**
- **MATLAB API**
 - interact with C and Fortran programs

Basic Matrix-Vector Operations

- Enter a matrix:

```
>>A=[ 3 2; 2 4]
```

- All matrices are enclosed in a pair of bracket [].
- Case-sensitive: matrix A and matrix a are different.

- Enter a vector

```
>>b = [4;3];
```

- b is a 2 x 1 column vector.
- The semicolon sign at the end of the line indicates that the interpreter will not echo that line on the screen.

- Matrix operations

```
>>c = A*b
```

```
>>AA = A^2
```

```
>>bb = b.^2
```

```
>>x = A\b
```

Basic Operators:

+	-	*	/	\	^	'
.*	./	.*	./	.\	.^	./

M-file Programming

- Use Matlab Editor create a program and save it as a m-file with a file-extension [.m]
- Simply enter the file name of the m-file (without the file extension) in the command window, the m-file will get executed.
- Example: use Matlab Editor to create a file called test.m. In test.m, enter the following lines

```
A = [3 2; 2 4]; b=[3; 2];
```

```
c = A*b;
```

```
A = A^2; b = b.^2;
```

```
x = A\b
```

Then in the command window, enter

```
>>test;
```

Function

- A function is a special m-file.
- A general syntax of a function:


```
function [outData1, outData2] =
myfunction(inData1, inData2, inData3)
```

Example:

```
function [c,x] = test_function(A,b)
```
- Workspace variables are not directly accessible to functions. They are normally passed to functions as arguments.
- All variable defined within a function are local variables. These variables will be erased after the execution of the function.

Visualization

2-D

plot Plot 2-D data with linear scales for both axes
loglog Plot with logarithmic scales for both axes

3-D

- Lines

plot3 Plot 3-D data with linear scales for all axes
contour Plot contour lines
quiver Plot vector fields

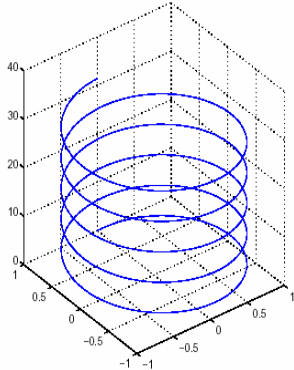
- Surfaces

mesh, surf: surface plot
meshc, surfc: surface plot with contour plot beneath it
surf1: surface plot illuminated from specified direction
surface: low-level function for creating surface graphics objects

Examples

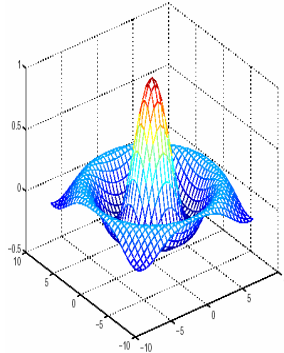
Line plots

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
axis square; grid on
```



Surface plots

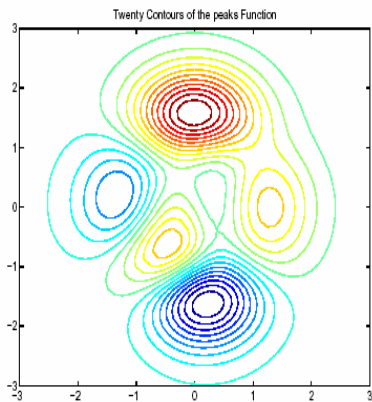
```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(Z)
```



More Examples

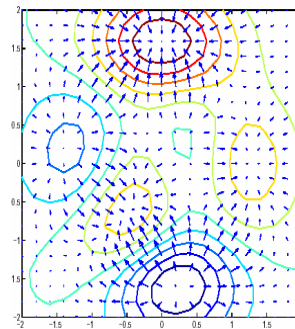
contour plots

```
[X,Y,Z] = peaks;
contour(X,Y,Z,20);
```



Vector fields

```
n = -2.0:.2:2.0;
[X,Y,Z] = peaks(n);
contour(X,Y,Z,10);
[U,V] = gradient(Z,.2);
hold on
quiver(X,Y,U,V)
```



Help

- Help on the toolbar

- Command line

```
>>help plot
>>helpwin plot
>>doc plot
```

- Demos

```
>>demo
```

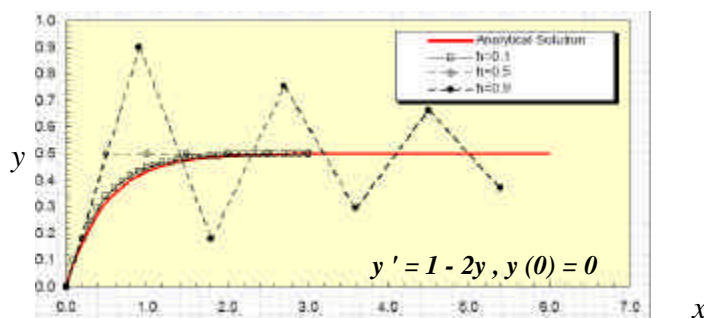
- Tutorial from mathworks:

http://www.mathworks.com/products/education/student_version/tutorials/launchpad.shtml

Numerical Integration of ODEs

$$\frac{dy}{dx} = f(x, y)$$

- Initial value problem: Given the initial state at $y_0=y(x_0)$, to compute the whole trajectory $y(x)$



Explicit Euler Solution

Euler's method

- **Explicit:** evaluate derivative using values at the beginning of the time step

$$y_{i+1} = y_i + h \times f(x_i, y_i) + O(h^2)$$

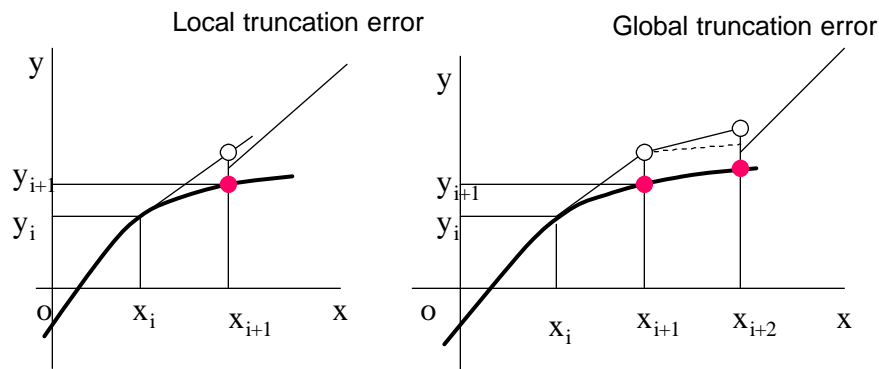
- Not very accurate (global accuracy $O(h)$) & requires small time steps for stability

- **Implicit:** Evaluate derivative using values at the end of the time step

$$y_{i+1} = y_i + h \times f(x_{i+1}, y_{i+1}) + O(h^2)$$

- May require iteration since the answer depends upon what is calculated at the end.
- Still not very accurate (global accuracy $O(h)$).
- Unconditionally stable for all time step sizes (why?).

Truncation errors



Stability

- A numerical method is **stable** if errors occurring at one stage of the process do not tend to be magnified at later stages.
- A numerical method is **unstable** if errors occurring at one stage of the process tend to be magnified at later stages.
- In general, the stability of a numerical scheme depends on the step size. Usually, large step sizes lead to unstable solutions.
- Implicit methods are in general more stable than explicit methods.

Stability Characteristics of Euler Methods

■ Model: $\frac{dy}{dx} = -ly; \quad y(0) = 1 \quad \text{D} \quad y(x) = 1 - e^{-lx}$
 $l > 0$

■ Error: $\frac{de}{dx} = -le$

- Explicit Euler Method is **conditionally stable**:

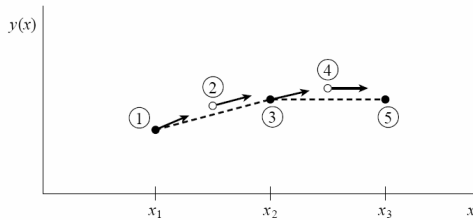
$$e_{i+1} = e_i + h(-le_i) = e_i(1 - lh) \quad \left| \frac{e_{i+1}}{e_i} \right| = |1 - lh| \leq 1 \quad \text{D} \quad 0 \leq h \leq \frac{2}{l}$$

- Implicit Euler Method is **unconditionally stable**:

$$e_{i+1} = e_i + h(-le_{i+1}) \quad \text{D} \quad e_{i+1} = \frac{e_i}{1 + lh}$$

$$\left| \frac{e_{i+1}}{e_i} \right| = \left| \frac{1}{1 + lh} \right| \leq 1 \quad \text{satisfied for all } h \geq 0$$

Second-order Runge-Kutta (midpoint method)



$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right)$$

$$y_{n+1} = y_n + k_2 + O(h^3)$$

- Second-order accuracy is obtained by using the initial derivative at each step to find a midpoint halfway across the interval, then using the midpoint derivative across the full width of the interval.
- In the above figure, filled dots represent final function values, open dots represent function values that are discarded once their derivatives have been calculated and used.
- A method is called n th order if its error term is $O(h^{n+1})$

Classic 4th-order R-K method

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right)$$

$$k_4 = f(x_i + h, y_i + hk_3)$$

where $h = x_{i+1} - x_i$ is the step size.

Local error is of $O(h^4)$. Global error is of $O(h^3)$.

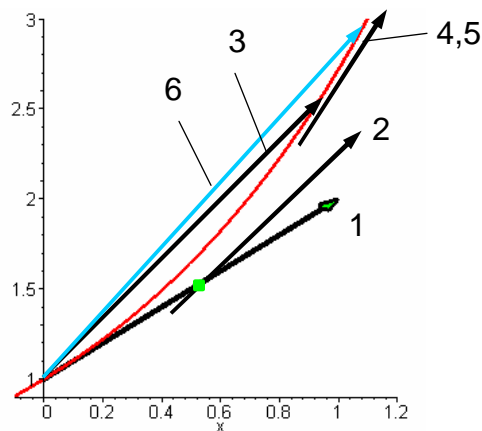
Runge-Kutta Methods

$$y(x+h) = y(x) + \sum_{i=1}^n w_i k_i$$

$$k_i = hf \left(x + c_i h, y(x) + \sum_{j=1}^{i-1} a_{i,j} k_j \right)$$

- c , a , and w are numerical coefficients chosen to satisfy certain conditions.
- n is the number of terms. For 4th order R-K method, $n = 4$

Runge-Kutta Methods



Runge-Kutta Methods

1. Make a first (tentative) step with the Euler method
2. Evaluate slope at intermediate point.
3. Use the adjusted slope and make a second (also tentative) step from the initial point.
4. Evaluate function at additional points and use this information to further adjust the slope to be used at *the start*
5. Evaluate function at as many other points as required and make further adjustments to the slope to be used at *the start*.
6. Combine all the estimates to make the actual step.

Multi-step Methods

- Runge-Kutta-methods are **one step methods**, only the current state is used to calculate the next state.
- **Multi-step methods** calculate the next value based on a time series of values, that might be from the past, or from the future.
 - **Explicit ($b_0 = 0$) & implicit** methods.
 - # of the previous steps.

$$y_{i+1} = a_1 y_i + a_2 y_{i-1} + \dots + h \times [b_0 f(x_{i+1}, y_{i+1}) + b_1 f(x_i, y_i) + b_2 f(x_{i-1}, y_{i-1}) + \dots]$$

- **Adams-Bashforth** Method (explicit $b_0 = 0$)
- **Admas-Moulton** Method (implicit)
- **Predictor-Corrector** Methods

Adams-Bashforth (A-B) Methods (Explicit)

$$y_{i+1} = y_i + \frac{h}{12} [23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}) + 5f(x_{i-2}, y_{i-2})]$$

for $i = 2, 3, \dots, n-1$.

$$a_1 = 1, \quad a_2 = 0, \quad b_0 = 0,$$

$$b_1 = 23/12, \quad b_2 = -16/12 \quad \& \quad b_3 = 5/12.$$

At the beginning, while y_0 is known,
 y_1 & y_2 are calculated using one-step method.
 The local error is of $O(h^4)$.

Adams-Moulton (A-M) Methods (Implicit)

$$y_{i+1} = y_i + \frac{h}{12} [5f(x_{i+1}, y_{i+1}) + 8f(x_i, y_i) - f(x_{i-1}, y_{i-1})]$$

for $i = 1, 2, 3, \dots, n-1$.

$$a_1 = 1, \quad a_2 = 0, \quad b_0 = 5/12,$$

$$b_1 = 8/12, \quad \text{and} \quad b_2 = -1/12.$$

y_0 is known, y_1 is calculated using one-step method. The local error is $O(h^4)$.

Predictor-Corrector Methods

Adams 3rd-order Predictor-Corrector Methods

- Predictor:** using 3rd-order **A-B three-step** method to predict y_{i+1}^P
 for $i = 2, 3, \dots, n - 1$.

$$y_{i+1}^P = y_i + \frac{h}{12} [23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}) + 5f(x_{i-2}, y_{i-2})]$$

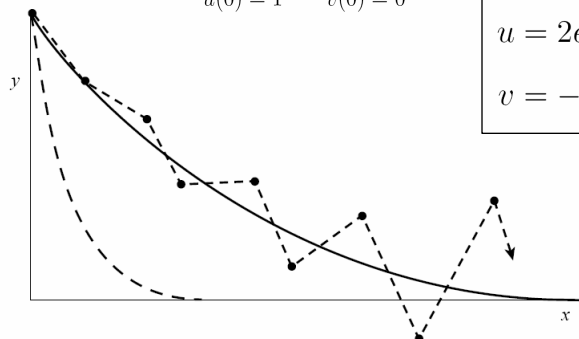
- Corrector:** using 3rd-order **A-M two-step** method to compute y_{i+1}^C

$$y_{i+1}^C = y_i + \frac{h}{12} [5f(x_{i+1}, y_{i+1}^P) + 8f(x_i, y_i) - f(x_{i-1}, y_{i-1})]$$

Stiff ODEs

- Example**

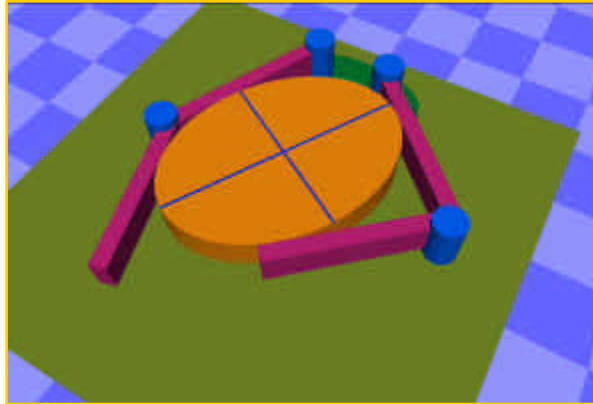
$$\begin{aligned} u' &= 998u + 1998v & u &= 2y - z & v &= -y + z \\ v' &= -999u - 1999v \\ u(0) &= 1 & v(0) &= 0 \end{aligned}$$



$$\begin{aligned} u &= 2e^{-x} - e^{-1000x} \\ v &= -e^{-x} + e^{-1000x} \end{aligned}$$

Stiff ODEs

- A practical example



Stiff ODEs

- Stiff systems are characterized by some system components which combine very fast and very slow behavior.
- Requires efficient step size control that adapt the step size dynamically, as only in certain phases they require very small step sizes.
- Implicit method is the cure!
 - Nonlinear systems: solving implicit models by linearization (**semi-implicit** methods)
 - Rosenbrock – generalizations of RK method
 - Bader-Deuffhard – semi-implicit method
 - Predictor-corrector methods – descendants of Gear's backward differentiation method

Higher Order ODEs

- Higher order ODEs can be turned into systems of 1st order ODEs:

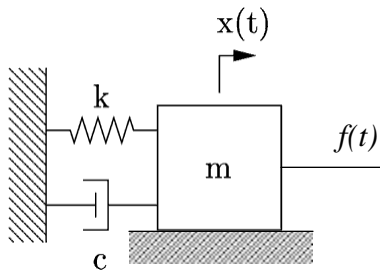
- For example:

$$\frac{d^3 y}{dx^3} + a \frac{d^2 y}{dx^2} + b \frac{dy}{dx} + cy = 2 \sin(x)$$

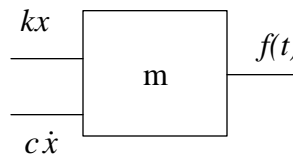
$$\mathbb{P} \begin{bmatrix} \frac{d}{dx} & 0 & 0 \\ 0 & \frac{d}{dx} & 0 \\ 0 & 0 & \frac{d}{dx} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \sin(x) \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & a & 0 \\ 0 & b & c \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

where $y_1 = y$ $y_2 = \frac{dy}{dx}$ $y_3 = \frac{d^2 y}{dx^2}$

Mass-Spring-Damper System



Free-body diagram



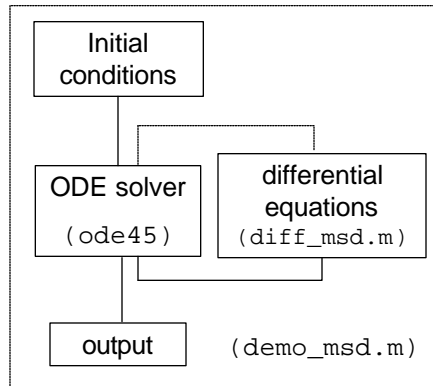
$$m \ddot{x} + c \dot{x} + kx = f(t)$$

$$\begin{aligned} y_1 &= x & \dot{y}_1 &= y_2 \\ y_2 &= \dot{x} & \dot{y}_2 &= -\frac{K}{m} y_1 - \frac{C}{m} y_2 + \frac{f(t)}{m} \end{aligned}$$

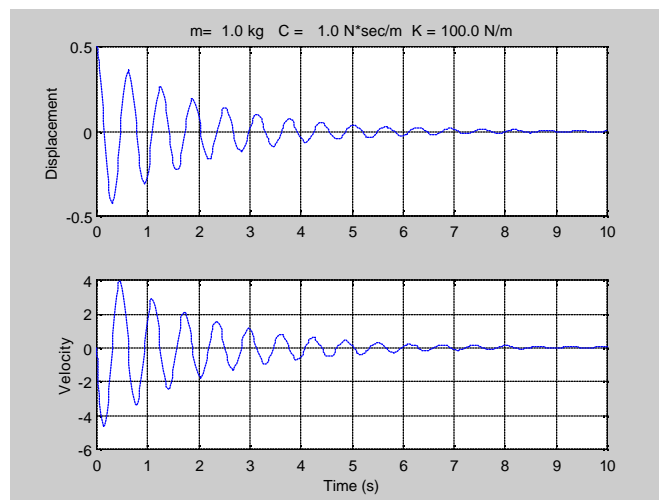
$$\dot{\mathbf{y}} = \begin{bmatrix} 0 & 1 \\ -K/m & -C/m \end{bmatrix} \mathbf{y} + \begin{bmatrix} 0 \\ f(t)/m \end{bmatrix}$$

Dynamic Simulation Example

Given $y_0=(x_0, 0)$, solve for $y(t)$ for $t=[0, T]$.



Simulation Results



Matlab Solvers for Nonstiff Problems

- `ode45`
 - Explicit Runge-Kutta (4,5) formula
 - One-step solver
 - Best function to apply as a "first try" for most problems
- `ode23`
 - Explicit Runge-Kutta (2,3)
 - One-step solver
 - May be more efficient than `ode45` at crude tolerances and in the presence of mild stiffness.
- `ode113`
 - Variable order Adams-Bashforth-Moulton PECE solver
 - Multistep solver
 - It may be more efficient than `ode45` at stringent tolerances and when the ODE function is particularly expensive to evaluate.

Matlab Solvers for Stiff Problems

- `ode15s`
 - Variable-order solver based on the numerical differentiation formulas (NDFs). Optionally it uses the backward differentiation formulas (BDFs).
 - Multistep solver.
 - If you suspect that a problem is stiff or if `ode45` failed or was very inefficient, try `ode15s` first.
- `ode23s`
 - Based on a modified Rosenbrock formula of order 2
 - One-step solver
 - May be more efficient than `ode15s` at crude tolerances
- `ode23t`
 - An implementation of the trapezoidal rule using a "free" interpolant
 - Use this solver if the problem is only moderately stiff and you need a solution without numerical damping

References

- Numerical Initial Value Problems in Ordinary Differential Equations, Gear, C.W., Englewood Cliffs, NJ: Prentice-Hall, 1971.
- Numerical Recipes in C : The Art of Scientific Computing William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, Cambridge University Press, 1992. (online at <http://www.library.cornell.edu/nr/bookcpdf.html>)