ME-GY 6923 Simulation Tools for Robotics
LECTURE 3

William Z. Peng, Ph.D.

NYU TANDON SCHOOL OF ENGINEERING

Numerical Methods, Algorithms, and Error Analysis / MATLAB - Part 2

ME-GY 6923 Simulation Tools for Robotics

LECTURE 3

William Z. Peng, Ph.D.

NYU TANDON SCHOOL OF ENGINEERING

Numerical Methods, Algorithms, and Error Analysis / MATLAB - Part 2

Lecture Overview

NYU TANDON SCHOOL OF ENGINEERING

- Last Lecture

  MATLAB – Numerical Methods, Algorithms, Solvers, and Error Analysis

  - Finite Difference Methods
    - Forward/Backward
    - 2-Step/Multi-step
    - Accuracy derivation for these methods

- This lecture

  We will look into some numerical methods, algorithms, and solvers (for solving differential equations)

- Next lecture

  We will begin our formal discussion of modeling using Simulink

Numerical Solutions of ODEs (Example 3-1)

Example of Initial Value Problem (IVP): Lotka-Volterra equation (predator-prey)

- In MATLAB, type "openExample('matlab/lotkademo')"

$$\begin{cases} R' = R - \alpha F R \\ F' = \beta R F - F \end{cases}$$

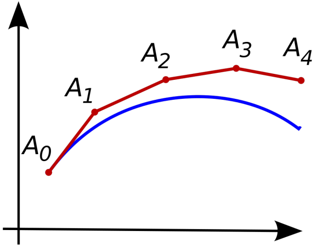
- $F(t)$ is the number of foxes (predator) at time t
- $R(t)$ is the number of rabbits (prey) at time t
- R' is the rate of growth of the rabbit population
    - R' increases with R but decreases with F (more foxes to eat the rabbits)
- F' is the rate of growth of the fox population
    - F' decreases F (food becomes scarce) but increases with R

No analytical solution exists – problem must be solved numerically

Euler Method - Numerical Algorithm for ODEs

NYU TANDON SCHOOL OF ENGINEERING

- Simplest algorithm (a.k.a forward Euler Method)
- We will look at a case where $n = 1$ (first order):
  $\dot{x} = f(x(t))$ $n = 1 \rightarrow$ one 1^{st} order ODE (Needs 1 initial condition)
- Method valid for case $n > 1$:
  $n > 1 \rightarrow n^{\text{th}}$ order ODE is converted into a system of n 1^{st} order ODEs. Needs n initial conditions
- Assumptions: uniform h and x has as many continuous derivatives as needed in order to analyze the accuracy of the methods using Taylor's theorem



See details here

Euler Method

NYU TANDON SCHOOL OF ENGINEERING

- One-step method: the one-step forward finite difference is used to approximate the time derivative of the solution at each node

One-step forward difference

$$\dot{x}(t_{\mathrm{k}}) = f(x(t_k)) \approx \frac{x(t_{k+1}) - x(t_k)}{h} \to f(x_k) = \frac{x_{k+1} - x_k}{h}$$

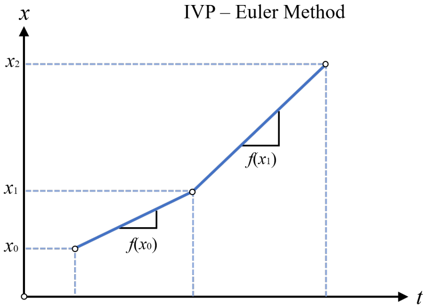
- Starting from IC $x_0 = x(t_0)$, the approximate solution at each time step is given by the iterative formula: $x_{k+1} = x_k + hf(x_k)$
- Evaluating this iterative formula step by step, for all $k = 0, \ldots, N$ ($N + 1 =$ number of total mesh points), we find the numerical solution to the ODE $\rightarrow$ Using the numerical method, the ODE is transformed into a set of algebraic equations

Qualitative Analysis:

Steps 1 – 4 describe the Euler Method algorithm:

1. Initial conditions: $x(t_0) = x_0$
2. From the D.E.: $\dot{x} = f(x_0)$
   - This gives the slope at the initial time t_0 and it is assumed constant for the entire h
3. From the previous step, the approximate solution at the next time-step $x_1 = x_0 + f(x_0)h$ may be found
4. From x_1, find $f(x_1)$, which is the new slope, from the D.E. and keep moving forward, one step at a time

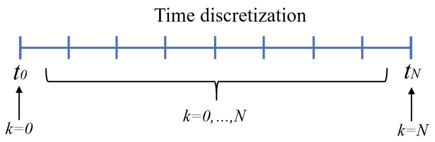
Note: Because the Euler Method is based on the one-step forward infinite difference method, it is also called a one-step method (whose error is $O(h)$)



D.E.: $\dot{x}(t) = f(x(t)), n = 1$

Euler Method - Accuracy

- Consider: $\mathbf{x}_E = [x_0, x_1, x_2, ..., x_N]$, where x_0 is known and $x_1, x_2, ..., x_N$ are N algebraic equations
- With: $x_{k+1} = x_k + f(x_k)h$, where $h = t_{k+1} - t_k$ can be evaluated for $k = 0, ..., N-1$ (N times)

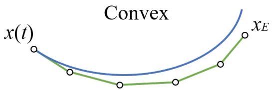


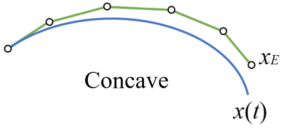
The time interval is initially discretized in $N+1$ nodes or N intervals

What can we say about the solution x_E?

- It is approximated, and
- It is either larger or smaller than the solution x(t) (which is not known)

For intervals of $x(t)$ that are concave and convex:





it may be qualitatively said that:

$$x_E < x(t)$$

if in that interval, $x(t)$ convex

$$x_E > x(t)$$

if in that interval, $x(t)$ concave

Euler Method - Example 3-2

IVP with Euler Method and qualitative argument about accuracy

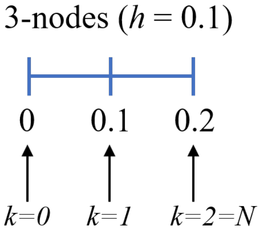
For: $x' = f(t, x)$

$$x' = t^2 - x^2$$

with the initial condition: $x(0) = x_0 = 1$

$$x_{k+1} = x_k + hf(t_k, x_k)$$

evaluate for $k = 0, 1, 2$:



- $k = 0$ $\quad x_1 = x_0 + hf(0, x_0) = 1 + 0.1 \cdot (0^2 - 1^2) = 1 - 0.1 \cdot 1 = 0.9$
- $k = 1$ $\quad x_2 = x_1 + hf(t_1, x_1) = 0.9 + 0.1 \cdot (0.1^2 - 0.9^2) = 0.9 - 0.08 = 0.82$
- $k = 2$ $\quad x_3 = x_2 + hf(t_2, x_2) = 0.82 + 0.1 \cdot (0.2^2 - 0.82^2) = 0.82 - 0.0632 = 0.757$

NODE	t	x	f	hf
0	0	1	-1	-0.01
1	0.1	0.9	-0.8	-0.08
2	0.2	0.82	-0.632	-0.0632

$$\mathrm{x}_E = \begin{bmatrix} 0.9 \\ 0.82 \\ 0.757 \end{bmatrix}$$

- Curvature: $x'' = 2t - 2xx'$
- At $t = 0$, $x_0 = 1$, $x_0' = f_0 = -1$ and $x_0'' = -2 \cdot 1 \cdot (-1) = 2 > 0$, so convex
- At $t = 1$, $x_1 = 0.9$, $x_1' = f_1 = -0.8$ and $x_1'' = 0.2 - 2 \cdot 0.9 \cdot (-0.8) = 1.64 > 0$, so still convex

Euler Method - Example 3-3

For: $y' = f(t, y)$

$$y' = y$$

with the initial condition: $y(0) = y_0 = 1$

$$y_{k+1} = y_k + hf(y_k)$$

we want to approximate $y(4)$:

- $k = 0$ $\quad y_1 = y_0 + hf(y_0) = 1 + 1 \cdot 1 = 2$
- $k = 1$ $\quad y_2 = y_1 + hf(y_1) = 2 + 1 \cdot 2 = 4$
- $k = 2$ $\quad y_3 = y_2 + hf(y_2) = 4 + 1 \cdot 4 = 8$
- $k = 3$ $\quad y_4 = y_3 + hf(y_3) = 8 + 1 \cdot 8 = 16$

The solution to the differential equation is: $y(t) = e^t$, so $y(4) = e^4 \approx 54.598$, while the Euler Method approximation result is 16. This is due to the very large step size

We can look at this in MATLAB and try different step sizes h

Euler Method - Accuracy

- Recall: truncation error comes from the finite difference approximation used in the current method. Recall: It is the amount by which the true solution fails to satisfy the current difference equation

- For: $x' = f(t, x)$

- By expanding the true solution with the Taylor's theorem:

$$
\begin{aligned}
x(t_{k+1}) &= x(t_k) + h\dot{x}(t_k) + \frac{h^2}{2}\ddot{x}(\xi_k) \\
&= x(t_k) + hf(t_k, x(t_k)) + \frac{h^2}{2}\ddot{x}(\xi_k) \text{ with } \xi \in [t_k, t_{k+1}]
\end{aligned}
$$

  which can be rewritten as: $\frac{x(t_{k+1})-x(t_k)}{h} = f(t_k, x(t_k)) + \frac{h}{2}\ddot{x}(\xi_k)$

- Compare with the finite difference used in Euler's method: $\frac{x_{k+1}-x_k}{h} = f(t_k, x(t_k))$

- Local truncation error is $+\frac{h^2}{2}\ddot{x}(\xi_k) \rightarrow$ Euler's method is order $O(h^2)$ (as expected $\rightarrow$ Forward Difference-based)

Euler Method - Accuracy

- The Euler algorithm is first order accurate. However, reducing the step size h comes at a price:
  - Increase the number of computation (mesh) points N
  - Increase cumulative error, due to roundoff:

$$\frac{x_{k+1}(1+\delta_1) - x_k(1+\delta_2)}{h} = \frac{x_{k+1} - x_k}{h} + \frac{x_{k+1}\delta_1 - x_k\delta_2}{h}$$

  with $|\delta_i| < \varepsilon_m$, an upper bound for roundoff can be found:

$$\frac{x_{k+1}\delta_1 - x_k\delta_2}{h} \leq \frac{|x_{k+1}|\,|\delta_1| + |x_k|\,|\delta_2|}{h} \leq \frac{|x_{k+1}|\,\varepsilon_m + |x_k|\,\varepsilon_m}{h} \leq \frac{\varepsilon_m\left(|x_{k+1}| + |x_k|\right)}{h} \propto \frac{\varepsilon_m}{h}$$

- As we saw before, the best trade off is when both errors are comparable: truncation $\sim$ roundoff
  $\longrightarrow h \approx \frac{\varepsilon_m}{h} \rightarrow h \approx \sqrt{\varepsilon_m}$ (for Euler Method)

Euler Method - Accuracy

- Local truncation error $O(h^2)$: the error that is contributed at each step (it's relative to one step of the Euler algorithm)
- Global truncation error: for the entire solution we evaluate N steps (when we have $N+1$ nodes) $\rightarrow$ global error $= N \times O(h^2) \rightarrow$ accumulated error $\rightarrow = O(h)$
- A method is called consistent if its local truncation error approaches 0 as h $\rightarrow$ 0 (impractical)
- Euler Method is convergent (for a well-posed problem), because it can be shown that As $h \rightarrow 0$, the max difference between x_{E} and $x(t) \rightarrow 0$, for any mesh point in the fixed interval of t (we won't show this)

Euler Method - Summary

In dynamical system simulation, the following precautions should be taken:

- Select a suitable step size: not too large nor too small
- The Forward Euler method is the simplest to implement but can become unstable and lead to inaccuracy
- Improved algorithms have been developed: Runge-Kutta, Adams, etc. These have higher order accuracy
- Because finding a suitable step size is very difficult in general, many ODE solvers support a variable step size method: when the error estimate$^{(*)}$ is large, smaller step size should be chosen, while when the error estimate is small, larger step size should be selected.

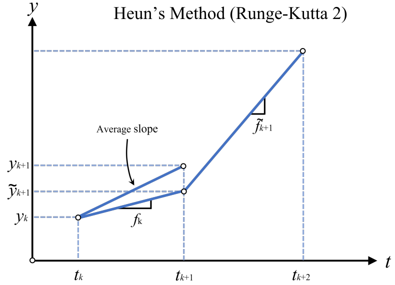
  $(*)$ Error estimate: Matlab solvers have a way of estimating local error (local = at every time step)

Multi-step method: Heun's method (Improved Euler's Method or Runge Kutta 2$^{\text{nd}}$ order)

- Graphical procedure: $y' = f(y(t))$
- Method: $y_{k+1} = y_k + h\left(\frac{f_k+\tilde{f}_{k+1}}{2}\right)$ - Eq. (i)
- where $\tilde{f}_{k+1}$ is evaluated using the standard Euler Method:

$$\tilde{f}_{k+1} = f(y_{k+1}, \tilde{y}_{k+1})$$

with $\tilde{y}_{k+1} = y_k + hf_k$ (standard Euler and intermediate step only to eliminate $\tilde{f}_{k+1}$)



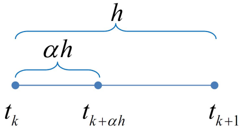
- It may be apparent that this is $O(h^2)$, since even though Eq. (i) looks like a 1-step method, this method is more similar to a 2-step method, given that there is an intermediate step
- By a similar reasoning, it is possible to further improve the order of accuracy as long as a new algorithm is delivered, that involves additional intermediate steps (which improve the estimate of the slope), leading to the fourth-order Runge-Kutta algorithm

Advanced Algorithms

NYU TANDON SCHOOL OF ENGINEERING

Classic 4th order Runge-Kutta :

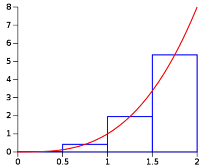
$$\tilde{y}_{k+\alpha} = y_k + \alpha h f(t_k, x_k)$$
$$y_{k+1} = y_k + \beta h f(t_k, x_k) + \gamma h f(t_k + \alpha h \tilde{y}_{k+1})$$



- Parameters α, β, γ must be chosen to match Taylor expansion of real solution with the finite difference iterative rule
- Doing so will yield: $y_{k+1} = y_k + h\frac{K_1+2K_2+2K_3+K_4}{6}$ where:

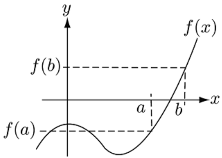
$$K_1 = f(t_k, y_k) \qquad K_3 = f(t_k + \tfrac{h}{2}, y_k + h\tfrac{K_2}{2})$$
$$K_2 = f(t_k + \tfrac{h}{2}, y_k + h\tfrac{K_1}{2}) \qquad K_4 = f(t_k + h, y_k + hK_3)$$

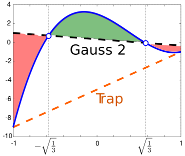
using the $O(h^4)$ method



Other methods include:

- Midpoint Rule
- Adams-Bashforth/Adams-Moulton
- Quadrature





Main commands and general rules of thumbs:

- ode45 is based on an explicit Runge-Kutta formula (4,5). It is a one-step solver, i.e., in computing $x(t_{k+1})$, it needs only the solution at the immediately preceding time point, $x(t_k)$. In general, is the best function to apply as a *first try* for most problems

- ode23 is an implementation of an explicit Runge-Kutta (2,3). It may be more efficient than ode45 at crude tolerances and with moderate stiffness

- ode113 is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than ode45 at stringent tolerances and when the ODE file function is particularly expensive to evaluate. ode113 is a *multistep* solver – it normally needs the solutions at several preceding time points to compute the current solution

MATLAB Solutions to ODEs

The aforementioned algorithms are intended to solve non-stiff systems. If they appear to be unduly slow, try using:

- ode15s is a variable order solver based on the numerical differentiation formulas (NDFs) or backward differentiation formulas (BDFs, also known as Gear's method)
- Try ode15s when ode45 fails, or is very inefficient, and you suspect that the problem is stiff, or when solving a differential-algebraic problem

Resources:

https://www.mathworks.com/help/matlab/ordinary-differential-equations.html?s_tid=srchtitle

https://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html

https://blogs.mathworks.com/cleve/2014/05/26/ordinary-differential-equation-solvers-ode23-and-

MATLAB Solutions to ODEs

The main syntax of the command:

Syntax

[t,x]=ode45(Fun,tspan,x_0,options,additional parameters);

- The format is consistent for all MATLAB "ode" solvers (e.g. ode23, ode45, etc.)
- Options can be accessed with the odeget() and odeset() commands
- Fun: must contain the descriptive function/functions
- Differential equation can be described in one of the following way:
  - Anonymous functions:
    y = @(t,x,additional parameters)The function content
  - M-functions:
    function x_1 = Fun(t,x,additional parameters)

Control Parameters of ODEs Solvers

options	parameter description
RelTol	relative error tolerance, with a default value of 0.001, i.e., the relative error is 0.1% in some applications, smaller values should be used.
AbsTol	absolute error tolerance, with a default value of 10^{-6}.
MaxStep	maximum allowed step size.
Mass	mass matrix in differential algebraic equations
Jacobian	Jacobian matrix $\partial \boldsymbol{f}/\partial \boldsymbol{x}$ function name; if the Jacobian matrix is known, the speed of computation is increased.

```
>> f=odeget; f.RelTol=1e-8;
```

http://www.mathworks.com/help/matlab/ref/odeset.html
http://www.mathworks.com/matlabcentral/answers/26743-absolute-and-relative-tolerance-definitions

Stiff ODEs

- The solution of certain systems changes very rapidly over certain time intervals, while it changes slowly in other parts
- Solution to these systems require a variable step size method
- If stiffness is relevant, requires particular ode solvers (ode45 won't work)
- In case of ODE of order > 1, we have stiff systems when the solutions of each state variable have very different range and behavior. Some variables change very fast and other change very slowly

Error Analysis

Quantitative error analysis:

- The solution's convergence can be observed qualitatively by looking at plots obtained with different values of h (e.g. see the Euler 2 MATLAB example)
- There are several quantitative analyses that can be done:
  - 1: Compare error vector of numerical vs. analytical solution (if any)
  - 2: Compare the error vector of various numerical solutions, obtained with different methods/tolerances

Error Analysis

Error vector of numerical vs. analytical:

- Assume the analytical solution is known, e.g.:

$$y(t) = \frac{1}{2} + e^t \left(-\frac{3}{20}\sin(t) + \frac{1}{2}\cos(t) + \frac{3}{20}t\cos(t) + \frac{1}{4}t\sin(t) \right)$$

- Implement in MATLAB
- Evaluate the function at the same mesh points t_k obtained from the numerical solution $[\mathbf{t}, \mathbf{y}_{\text{num}}]$ with dimension $[N \times 1]$. This means, substitute variable t with vector
- Calculate the 2-norm of error vector $\mathbf{e} = \mathbf{y}(t_k) - \mathbf{y}_{\text{num}}$, with $k = 1, ..., N$

$$\parallel e \parallel = \sqrt{e_1^2 + e_2^2 + ... + e_N^2}$$

Conversion of ODE sets

- Convert the differential equation of other forms to the standard form

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x})$$

- Converting a single high-order ODE

$$y^{(n)} = f\left(t, y, \dot{y}, ..., y^{(n-1)}\right)$$

- A set of state variables can be selected as:

$$x_1 = y, x_2 = \dot{y}, ..., x_n = y^{(n-1)}$$

Conversion of ODE sets

The original high-order ODE can be converted to the following standard form:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \vdots \\ \dot{x}_n = f(t, x_1, x_2, ..., x_n) \end{cases} \qquad \begin{cases} x_1(0) = y(0) \\ x_2(0) = \dot{y}(0) \\ \vdots \\ x_n(0) = y^{(n-1)}(0) \end{cases}$$

Converting High-Order ODE sets

The ODE set:

$$\begin{cases} x^{(m)} = f(t, x, \dot{x}, ..., x^{(m-1)}, y, ..., y^{(n-1)}) \\ y^{(n)} = g(t, x, \dot{x}, ..., x^{(m-1)}, y, ..., y^{(n-1)}) \end{cases}$$

Select the state variables:

$$\begin{cases} x_1 = x \\ \vdots \\ x_m = x^{(m-1)} \\ x_{m+1} = y \\ \vdots \\ x_{m+n} = y^{(n-1)} \end{cases} \qquad \begin{cases} \dot{x}_1 = x_2 \\ \vdots \\ \dot{x}_m = f(t, x_1, x_2, ..., x_{m+n}) \\ \dot{x}_{m+1} = x_{m+2} \\ \vdots \\ \dot{x}_{m+n} = g(t, x_1, x_2, ..., x_{m+n}) \end{cases}$$

Solutions to DAEs

- In certain differential equations, some of the state variables satisfy certain algebraic constraints

$$\mathbf{M}(t, \mathbf{x})\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x})$$

- Mass Matrix in general non-singular
- Set the Mass options. `Mass = M`

Example (see Exercise 4-4, below):

$$\begin{cases} \dot{x}_1 = -0.2x_1 + x_2x_3 + 0.3x_1x_2 \\ \dot{x}_2 = 2x_1x_2 - 5x_2x_3 - 2x_2^2 \\ x_1 + x_2 + x_3 - 1 = 0 \end{cases}$$

with some initial conditions

MATLAB Exercise 3-1

M-function vs. Anonymous function:

Mathematical functions can be described in MATLAB in one of the following ways:
M-function vs. Anonymous function (Example 1)

$$\dot{y}_1 = y_2 y_3 \quad \dot{y}_2 = -y_1 y_3 \quad \dot{y}_3 = -0.51 y_1 y_2$$

with initial values $y_1 = 0, y_2 = 1, y_3 = 1$, and total time duration $t = [0, 12]$

Write a script "Exercise3_1.m"*:

- Follows Example 1 in MATLAB ode45 documentation
- Use ODE 45
- Set tolerance
- Validate results
- Plot step size

* Reminder: Matlab will not run a file that contains a "-" in the filename, so use a "_" instead

MATLAB Exercise 3-2

Van der Pol:

Van der Pol second order non linear differential equations:

$$\ddot{y} + \mu(y^2 - 1)\dot{y} + y = 0$$

Convert to standard form:

$$\mathbf{x} = [x_1, x_2] = [y, \dot{y}] \rightarrow \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\mu(x_1^2 - 1)x_2 - x_1 \end{bmatrix}$$

with initial values $\mathbf{x}_0 = [-0.2, -0.7]$ and total time duration 20s

Write a script "Exercise3_2.m":

- Use ode45, with parameter μ
- Try different implementation of ODE descriptive function (anonymous and M-function)
- Visualize time responses and solution in the state space
- Stiff problem $\rightarrow$ see when ode45 fails

MATLAB Exercise 3-3

Apollo satellite:

Satisfies the following ODE:

$$\begin{cases} \ddot{x} = 2\dot{y} + x - \frac{\mu^*(x+\mu)}{r_1^3} - \frac{\mu(x-\mu^*)}{r_2^3} \\ \ddot{y} = -2\dot{x} + y - \frac{\mu^* y}{r_1^3} - \frac{\mu y}{r_2^3} \end{cases}$$

$$\mu = 1/82.45 \qquad \mu^* = 1 - \mu$$

$$r_1 = \sqrt{(x_1 + \mu)^2 + x_3^2} \qquad r_2 = \sqrt{(x_1 + \mu^*)^2 + x_3^2}$$

$$x(0) = 1.2, \quad \dot{x}(0) = 0, \quad y(0) = 0, \quad \dot{y}(0) = -1.04935751$$

Select the state variables: $x_1 = x$, $x_2 = \dot{x}$, $x_3 = y$, $x_4 = \dot{y}$

MATLAB Exercise 3-3

Apollo satellite:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = 2x_4 + x_1 - \mu^*(x_1+\mu)/r_1^3 - \mu(x_1-\mu^*)/r_2^3 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = -2x_2 + x_3 - \mu^* x_3/r_1^3 - \mu x_3/r_2^3 \end{cases}$$

$$\mu = 1/82.45 \qquad \mu^* = 1 - \mu$$

$$r_1 = \sqrt{(x_1+\mu)^2 + x_3^2} \qquad r_2 = \sqrt{(x_1+\mu^*)^2 + x_3^2}$$

```
function dx=apolloeq(t,x)
mu=1/82.45; mu1=1-mu;
r1=sqrt((x(1)+mu)^2+x(3)^2); r2=sqrt((x(1)-mu1)^2+x(3)^2);
dx=[x(2);
    2*x(4)+x(1)-mu1*(x(1)+mu)/r1^3-mu*(x(1)-mu1)/r2^3;
    x(4);
    -2*x(2)+x(3)-mu1*x(3)/r1^3-mu*x(3)/r2^3];
```

MATLAB Exercise 3-4

DAEs:

$$\begin{cases} \dot{x}_1 = -0.2x_1 + x_2x_3 + 0.3x_1x_2 \\ \dot{x}_2 = 2x_1x_2 - 5x_2x_3 - 2x_2^2 \\ x_1 + x_2 + x_3 - 1 = 0 \end{cases}$$

The standard form of a DAE:

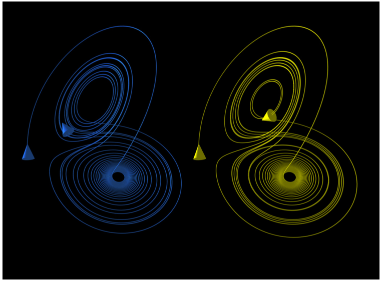
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -0.2x_1 + x_2x_3 + 0.3x_1x_2 \\ 2x_1x_2 - 5x_2x_3 - 2x_2^2 \\ x_1 + x_2 + x_3 - 1 \end{bmatrix}$$

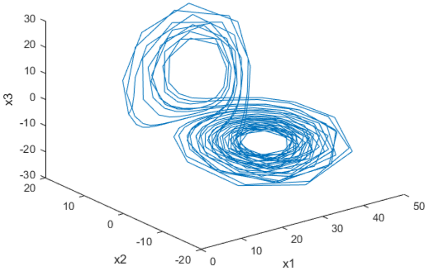
MATLAB commands

```
>> f=@(t,x)[-0.2*x(1)+x(2)*x(3)+0.3*x(1)*x(2);
         2*x(1)*x(2)-5*x(2)*x(3)-2*x(2)*x(2);
         x(1)+x(2)+x(3)-1];
   M=[1,0,0; 0,1,0; 0,0,0]; options=odeset;
   options.Mass=M; x0=[0.8; 0.1; 0.1];
   [t,x]=ode15s(f,[0,20],x0,options); plot(t,x)
```

Lorenz Attractor:

$$\begin{cases} \dot{x}_1 = 8x_1(t)/3 + x_2(t)x_3(t) \\ \dot{x}_2 = -10x_2(t) + 10x_3(t) \\ \dot{x}_3 = -x_1(t)x_2(t) + 28x_2(t) - x_3(t) \end{cases}$$





Homework and References

Assignment

- Study Examples 3-1 – 3-3 and Exercises 3-1 – 3-5 (no need to submit)
- Complete the Simulink Onramp Course
  https://www.mathworks.com/learn/tutorials/simulink-onramp.html
- Upload Completion Report Certificate (as a pdf) **and** the Link: Once you have completed the course, select "View/Share Certificate" on the "My Self-Paced Courses" section of the Mathworks website, and post the "shareable link" to your Progress Report to Brightspace (ensure that the link works)

References

- Xue & Chen Chapter 3: 3.4.1 – 3.4.5