

# Unit 4: Numerical analysis using MATLAB



## Introduction

Dynamic behavior of systems involve either mechanical system of displacements, velocities, and accelerations forms or electronic systems of voltages, currents, and time derivatives of these quantities. These systems are mathematically described by an equation that involves one or more derivatives of the unknown function called an ordinary differential equation, abbreviated as ODE. Subject to the order of the highest derivative, ODEs are classified as either first-order ODE (i.e. highest derivative is first order), or second-order ODE (highest derivative is second order) or n-th order ODE (highest derivative is n-th order). These equations can either have exact or analytical solutions, but in most cases they appear too complex to solve by hand. Therefore, in this Unit, we look at how to use MATLAB to solve such systems of differential equations using either numerical differentiation or numerical integration methods. Specifically we shall focus on methods that belong to a family of Runge-Kutta (RK) such as Euler methods.

---



## Unit Objectives

On successful completion of the Unit, students should be able to:

- Understand process of programming RK-numerical methods for solving ODES
  - Apply MATLAB numerical methods to solve related real life problems
- 



## Key Terms

As you go through this unit, ensure that you understand the key terms or phrases used in this unit as listed below:

- ODE
- Euler
- Dependent and independent variable
- Runge-Kutta
- Trapezoid
- Simpsons



## Error analysis

In computational mathematics, we have different sources of errors, the major ones being;

- **Truncation errors:** These result from the premature termination of an infinite computation.
- **Round off errors:** These result from using floating point arithmetic. Less significant than truncation errors, but nevertheless can result in catastrophic problems
- **Other errors:** For other errors, we account for: Human errors, modeling errors, and measurement errors.

But in order to measure these types of errors; we have two ways of doing so. If we let  $p$  be an approximation to  $p^*$ , then we measure the errors by defining

- **Absolute error**

$$|p - p^*|$$

- **Relative error**

$$\frac{|p - p^*|}{|p^*|}$$



## Numerical differentiation

Here, we discuss numerical methods for solving ODEs using Euler's methods, namely forward Euler, modified Euler and backward Euler.

### 1. Forward Euler

Given a differential equation of the form  $y' = f(x, y)$  where  $y' = \frac{dy}{dx}$ , the method involves re-writing the forward difference approximation

$$y'_n = \frac{y_{n+1} - y_n}{h}$$

to

$$y_{n+1} = y_n + hy'_n$$

But since  $y' = f(x, y)$ , then this re-writing translates into

$$y_{n+1} = y_n + hf(x_n, y_n)$$

Thus, for  $i = 0$  to  $n$ , we can recursively obtain the generalized forward Euler method as

$$\begin{aligned} y_1 &= y_0 + hy'_0 \\ y_1 &= y_0 + hf(x_0, y_0) \\ y_2 &= y_1 + hf(x_1, y_1) \\ y_3 &= y_2 + hf(x_2, y_2) \\ &\vdots \\ y_n &= y_{n-1} + hf(x_{n-1}, y_{n-1}) \end{aligned}$$

Other than forward Euler, we have the modified version of it, which is better than the standard Euler method.

## 2. Modified Euler

The modified Euler method is better than forward Euler because it is more accurate than the forward Euler method and is also more stable. We can derive the modified Euler by applying the trapezoidal rule to the solution of  $y' = f(x, y)$ . Thus, we have

$$y_{n+1} = y_n + \frac{h}{2}[f(x_{n+1}, y_{n+1}) + f(x_n, y_n)]$$

## 3. Backward Euler

Then, with similarity in accuracy with forward Euler, we have the backward Euler which is based on backward differencing. Hence, we have

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1})$$

In general, all Euler methods follow the same similar approach when coding them in MATLAB. Such an approach can be generalized into a form of ***pseudo-code*** as follows

```
Step 1. define  $f(x, y)$ .  
Step 2. input initial values  $x_0$  and  $y_0$   
Step 3. input step sizes  $h$  and number of steps  $n$   
Step 4. calculate  $x$  and  $y$ :  
for  $i = 1:n$   
     $x = x + h$   
     $y = y + hf(x, y)$   
end  
Step 5. output  $x$  and  $y$   
Step 6. end
```

### Example

Let us consider a differential equation with an initial value

$$\frac{dy}{dx} = \frac{x}{y}, y(0) = 1$$

whose analytical solution is  $y(x) = \sqrt{x^2 + 1}$

However, going by forward Euler's method, we have

<pre>function [x,y]=euler_forward(f,x0,y0,xf,n) % Euler approximation for ODE initial value problem % Euler forward method % Calculation of h from x0, xf, and n h=(xf-x0)/n; % Initialization of x and y as column vectors x=[x0 zeros(1,n)]; y=[y0 zeros(1,n)]; % Calculation of x and y for i=1:n     x(i+1)=x(i)+h;     y(i+1)=y(i)+h*f(x(i),y(i)); end end</pre>	<pre>&gt;&gt; f=@(x,y) x./y; &gt;&gt; [x,y]=euler_forward(f,0,1,0.3,6)  x =     0    0.0500    0.1000    0.1500    0.2000 0.2500    0.3000  y =     1.0000    1.0000    1.0025    1.0075 1.0149    1.0248    1.0370</pre>
---	---

Alternatively, going by the modified Euler's method, we have

<pre>function [x,y]=euler_modified(f,x0,y0,xf,n) % Euler approximation for ODE initial value problem % Euler modified method % Calculation of h from x0, xf, and n h=(xf-x0)/n; % Initialization of x and y as column vectors x=[x0 zeros(1,n)]; y=[y0 zeros(1,n)]; % Calculation of x and y for i=1:n     x(i+1)=x(i)+h;     ynew=y(i)+h*f(x(i),y(i));  y(i+1)=y(i)+(h/2)*(f(x(i),y(i))+f(x(i+1),ynew)); end end</pre>	<pre>&gt;&gt; f=@(x,y) x./y; &gt;&gt; [x,y]=euler_modified(f,0,1,0.3,6)  x =     0    0.0500    0.1000    0.1500 0.2000    0.2500    0.3000  y =     1.0000    1.0013    1.0050    1.0112 1.0198    1.0308    1.0440</pre>
---	--

And then with backward Euler

<pre>function [x,y]=euler_backward(f,x0,y0,xf,n) % Euler approximation for ODE initial value problem % Euler backward method % Calculation of h from x0, xf, and n h=(xf-x0)/n; % Initialization of x and y as column vectors x=[x0 zeros(1,n)];</pre>	<pre>&gt;&gt; f=@(x,y) x./y; &gt;&gt; [x,y]=euler_backward(f,0,1,0.3,6)  x =     0    0.0500    0.1000    0.1500    0.2000 0.2500    0.3000  y =</pre>
--	--

<pre> y=[y0 zeros(1,n)]; % Calculation of x and y for i=1:n     x(i+1)=x(i)+h;     ynew=y(i)+h*(f(x(i),y(i)));     y(i+1)=y(i)+h*f(x(i+1),ynew); end end </pre>	<pre> 1.0000 1.0025 1.0075 1.0149 1.0247 1.0367 1.0511 </pre>
---	---

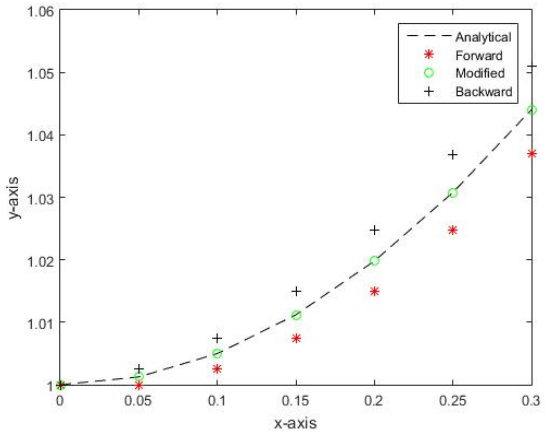
These three solutions can then be compared with the exact or analytical solutions obtained via the formula

$$y(x) = \sqrt{x^2 + 1}$$

In MATLAB, we have

<pre> function ye=exact_soln % calculate exact solution of the given function x=[0:0.05:0.3]; g=@(x) sqrt(x.^2+1); ye=g(x); end </pre>	<pre> &gt;&gt; exact_soln  ans =      1.0000    1.0012    1.0050    1.0112     1.0198    1.0308    1.0440 </pre>
--	--

Now, if we define a multiple plot function to compare the solutions of forward Euler, modified Euler and backward Euler with those of exact solution, we get the following output

<pre> function multiplot % compare the performance of Euler methods with analytical solution to % solving the ODE dy/dx=x/y f=@(x,y) x./y; % an ODE g=@(x) sqrt(x.^2+1); % an exact solution xe=[0:0.05:0.3]; x0=0; y0=1; xf=0.3; n=6; % call euler functions [x1,y1]=euler_forward(f,x0,y0,xf,n); [x2,y2]=euler_modified(f,x0,y0,xf,n); [x3,y3]=euler_backward(f,x0,y0,xf,n); % call analytical function [ye]=exact_soln; % plot functions plot(xe,ye,'k--',x1,y1,'r*',x2,y2,'go',x3,y3,'k+'); xlabel('x-axis'); ylabel('y-axis'); legend('Analytical','Forward','Modified', 'Backward'); end </pre>	 <p>Figure 1: Comparison of ODE numerical techniques</p>
---	--

Comparably, according to the results (Figure 1), forward and backward approaches give identically the same results, while modified method gives a very good result when compared with the exact solution, i.e. almost exactly equal to the analytical solution.

Apart from writing our own codes to solve ODEs, MATLAB has several different built-in functions for the numerical solution of not only ODEs but also partial differential equations (PDEs). In this unit, we would like to present one of them, however and possibly also give an example how to use it. This function is a RK-based called *odesolve* with the basic steps, as previously defined. These solvers can be generally used with the following syntax:

$$[x, y] = \text{solver}(@\text{odefun}, \text{tspan}, y_0)$$

To explain the syntax, *solver* is the solver we shall be using, such as *ode45* or *ode23*. Then, *odefun* is the function that defines the derivatives. So in the given syntax, *odefun* defines  $y_{\text{initial}}$  as a function of the independent parameter (typically  $x$  or  $t$ ) as well as  $y$  and other parameters. On the other hand, *tspan* a vector that specifies the interval of the solution (e.g.,  $[t_{\text{initial}}, t_{\text{final}}]$ ).  $y_{\text{initial}}$  is the initial value of  $y$ . Lastly,  $[x, y]$  is the output, which is the solution of the ODE.

We therefore present and discuss a general outline of the classical fourth-order Runge-Kutta methods. Given

$$y' = f(x, y), \quad y(x_n) = y_n$$

We compute the terms  $K_i$ 's such that

$$\begin{aligned} K_1 &= hf(x_n, y_n) \\ K_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{K_1}{2}\right) \\ K_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{K_2}{2}\right) \\ K_4 &= hf(x_n + h, y_n + K_3) \\ y_{n+1} &= y_n + \frac{1}{6}[K_1 + 2K_2 + 2K_3 + K_4] \end{aligned}$$

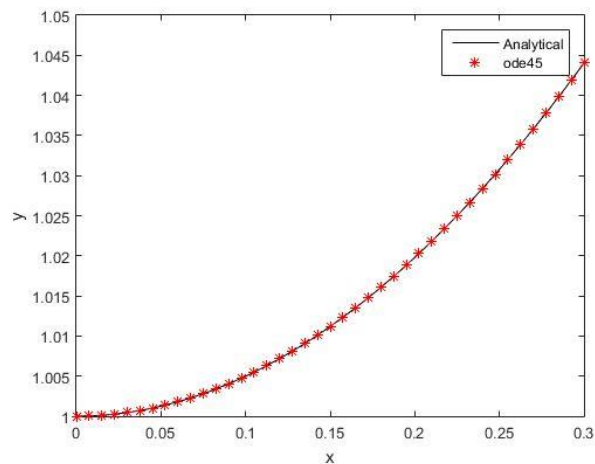
One of the key solvers used for solving system of ODEs in MATLAB is an *ode45* solver which combines both the fourth and fifth-order RK methods, having the syntax

$$\text{ode45}(@\text{xdot}, \text{tspan}, y_0)$$

Unlike with other RK methods, the solver ode45 is suitable for a wide variety of initial value problems (IVP) in practical applications. The modified RK varies the step size, choosing the step size at each step in an attempt to achieve the desired accuracy. In general, ode45 is the best function to apply as a “first try” for most problems.

For example, for the differential equation

```
function RK_method
f=@(x,y) x./y;
% Calculate exact solution
g=@(x) sqrt(x.^2+1);
xe=[0:0.01:0.3];
ye=g(xe);
% Call function
[x4,y4]=ode45(f,[0,0.3],1);
% Plot
plot(xe,ye,'k-',x4,y4,'r*')
xlabel('x')
ylabel('y')
legend('Analytical','ode45')
axis([0 0.3 1 1.05])
end
```



Based on the outputs, similar to the modified Euler, we observe that the *ode45* performs quite well as in approximating the solution to the differential equation

$$\frac{dy}{dx} = \frac{x}{y}, y(0) = 1$$

since the ode45 solution graph is exactly the same as that of the exact solution. Additional solvers that can be found in the online MATLAB help are as well presented in Table 1.

Table 1: Some MATLAB ode solvers

Solver	Description	Accuracy
<i>Ode45</i>	Most preferred and easy to implement	Medium
<i>Ode23</i>	Less accurate than ode45	Low
<i>Ode113</i>	Better for computationally expensive problems	Low to High
<i>Ode15s</i>	May be used if ode45 fails	Low to medium





## Numerical integration

The problem of integration is stated as, “Given a continuous function  $f(x)$  over an interval  $[a, b]$ , a numerical approximation  $I = \int_a^b f(x)dx$ ” can be estimated. In numerical integration, several techniques of approximating such integrals exist, but in this module we shall only introduce four of these, namely; Mid-point Rule, Trapezoid Rule, Simpson’s Rule and composite Simpson’s Rule

### a) Mid-point Rule

By the mid-point rule, we choose  $y(x)$  constant and sample at  $x_0 = (a + b)/2$ . We obtain  $y(x) = f_0 = f\left(\frac{a + b}{2}\right)$ . The numerical approximation is then computed as

$$I_0 = \int_a^b f\left(\frac{a + b}{2}\right) dx = (b - a) f\left(\frac{a + b}{2}\right)$$

### b) Trapezoid Rule

With Trapezoid rule, consider  $y(x)$  to be a linear function between the endpoints of the interval, such that  $(x_0 = a, f_0 = f(x_0))$  and  $(x_1 = b, f_1 = f(x_1))$ . Thus,  $y(x)$  can now be written as an interpolating polynomial in Lagrange form as

$$y(x) = \frac{(x - x_1)}{(x_0 - x_1)} f_0 + \frac{(x - x_0)}{(x_1 - x_0)} f_1$$

Thus, the numerical approximation using the interpolating polynomial is then defined as

$$\begin{aligned} I_1 &= \int_a^b \left[ \frac{(x - x_1)}{(x_0 - x_1)} f_0 + \frac{(x - x_0)}{(x_1 - x_0)} f_1 \right] dx \\ &= \frac{f(a)}{a - b} \left( \frac{-(a - b)^2}{2} \right) + \frac{f(b)}{b - a} \left( \frac{(b - a)^2}{2} \right) \end{aligned}$$

Such that we obtain the trapezoid rule as

$$I_1 = (b - a) \frac{1}{2} [f(a) + f(b)]$$

### c) Simpson’s Rule

For the Simpson's rule, we choose  $y(x)$  of order or degree 2, to be a parabola within the interval. Such that the interpolated points now become  $(x_0 = a, f_0 = f(x_0))$ ,  $(x_1 = (a + b)/2, f_1 = f(a + b/2))$  and  $(x_2 = b, f_2 = f(x_2))$ .

Such that we obtain the Simpson's rule as

$$I_2 = [w_0 f_0 + w_1 f_1 + w_2 f_2]$$

Where

$$w_0 = \int_{x_0}^{x_2} \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} dx = \frac{b - a}{6}$$

$$I_2 = (b - a) \frac{1}{6} [f_0 + 4f_1 + f_2]$$

#### 1. Example

Consider the approximation of an integral  $\int_a^b f(x)dx$  to within an error tolerance  $\varepsilon > 0$ . The Simpson's rule approximates this integral as  $\int_a^b f(x)dx = S(a, b) - \frac{h^5}{90} f^{(4)}(\mu)$  for  $\mu \in (a, b)$ ,  $h = \frac{b-a}{2}$ . Write the explicit expression for  $S(a, b)$ . By considering the composite Simpson's rule with 4 subdivisions, show that this approximation is 15 times better than the Simpson's rule  $S(a, b)$ .

#### d) Composite Simpson's Rule

##### 1. Example

The integral  $\int_a^b f(x)dx$  can be approximated more accurately using the composite Simpson's rule on  $n$  subintervals as:  $\int_a^b f(x)dx \approx \frac{h}{3} \left[ f(x_0) + 2 \sum_{j=1}^{\frac{n}{2}-1} f(x_{2j}) + 4 \sum_{j=1}^{\frac{n}{2}} f(x_{2j-1}) + f(x_n) \right]$ . For  $f(x) = \ln(\sqrt{1+x^2})$ ,  $a = 0$ ,  $b = 2$ , and  $n = 10$ , complete (lines 7 to 12) the following MATLAB code that can be used to implement the composite Simpson's rule;

# Unit 5: Mathematical Computing in Real Life

## Applications: Case Studies



### Introduction

Mathematical computing involves mathematical research in areas of science where computing plays a central and essential role, emphasizing algorithms, numerical methods, and symbolic computations. In this area of mathematics, much emphasis is given on using computers and calculators as tools to solve mathematically modeled physical problems. The result is a method that integrates mathematical theory into solutions for real-world problems, offering the best of both worlds: mathematics and computations. This unit of Mathematical Computing in Real-life Applications provides an introductory session of **four case studies** using different computational techniques for various real life applications. We shall observe that the techniques are simply derived from the previous four units are applied to solve

---



### Unit Objectives

On successful completion of the Unit, students should be able to:

- Translate real life problems into mathematical equations
  - Apply the mathematical computing techniques to solve related real life problems
  - Solve industrial problems as derived from different science and social science disciplines
- 



### Key Terms

As you go through this unit, ensure that you understand the key terms or phrases used in this unit as listed below:

- ODE
  - SIR
  - Exponential growth/decay
  - Model
  - Function file
  - Probability density function/ Cumulative distribution function
-



## Modeling infectious diseases

The field of mathematical computing is also more valuable in modeling infectious diseases by use of simulating disease dynamics using a set of differential equations. In this section, we shall demonstrate how to model disease dynamics of an epidemic, with the help of a set of differential equations and simulate the results using MATLAB. Attention will be paid much onto the design of the model, deduction of the differential equations, and solving using numerical methods and not analytically. For the theory and analytical models, students will have a chance to go through provided references.

Several disease models exist, ranging from SI models to SEIR models, where SI-means Susceptible-Infected population model; SIR-Susceptible-Infected-Recovered population; SIS-Susceptible-Infected-Susceptible population; and SEIR-Susceptible-Exposed-Infected-Recovered population. Further, these compartmental models are grouped into either deterministic or stochastic subject to underlying assumptions of disease and population dynamics.

### Example: SIR model

Let's suppose we have a population of individuals  $N$  who may be infected by some virus to become an infected population  $I$  and later some recover to become a recovered population  $R$ . Of course, it means some portion of the population may die, either naturally or due to the viral disease. Further, this population of individuals, at onset, assuming only one person was initially infected, then the remaining  $N - 1$  population becomes a susceptible to infection population denoted by  $S$ . Thus, we have the following diagram of representation for such

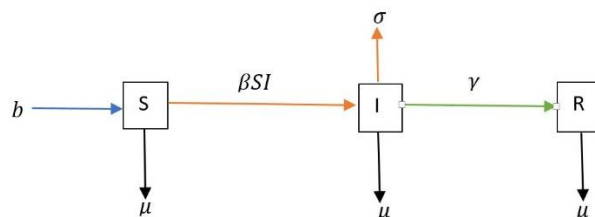


Figure 2: SIR model diagram with demographics

Where

- $b$  – birth rate
- $\mu$  – natural death rate
- $\sigma$  – death rate due to disease or infection
- $\beta$  – interaction rate
- $\gamma$  – recovery rate

Such that a population of size  $N$ , we have from Figure 2, the following set of ordinary differential equations

$$\begin{aligned}\frac{dS}{dt} &= bN - \beta SI - \mu S \\ \frac{dI}{dt} &= \beta SI - I(\mu + \sigma + \gamma) \\ \frac{dR}{dt} &= \gamma I - \mu R\end{aligned}$$

We shall therefore solve for the system of ODE to describe the disease dynamics using Excel and then numerically using the ode solver in MATLAB.

### Solution (a) SIR model in Excel

For our SIR model, we shall simulate with the following parameters

Total population	1000	
Infection Rate	0.2	20
Interaction Rate	0.0015	15
Initial Infected	1	
Delta Time	1	
Recovery Rate	0.1	

### Starting the Model

1. Label cell A4:D4: **Time**, **Susceptible**, **Infected** and **Recovered**. These columns will keep track of the population over time. Each row will represent another time step as the disease progresses.
2. In G4:G9, add labels for the following constants: Total, Infection rate, Interaction rate, Initial infected, Delta time, Recovery rate
3. In H4:H9, enter the following values for these numbers: *total*(1000), *infection\_rate*(I5/100), *interaction\_rate*(I6/10000), *initial\_infected*(1), *delta\_time*(1), *recovery\_rate*(0.1). Cell I5=20, I6=15.
4. Initialize: A5=0, B5= total - initial\_infected, C5= initial\_infected, D5=0;
5. **Writing the equations:**

**Time** A6 = A5 + delta\_time

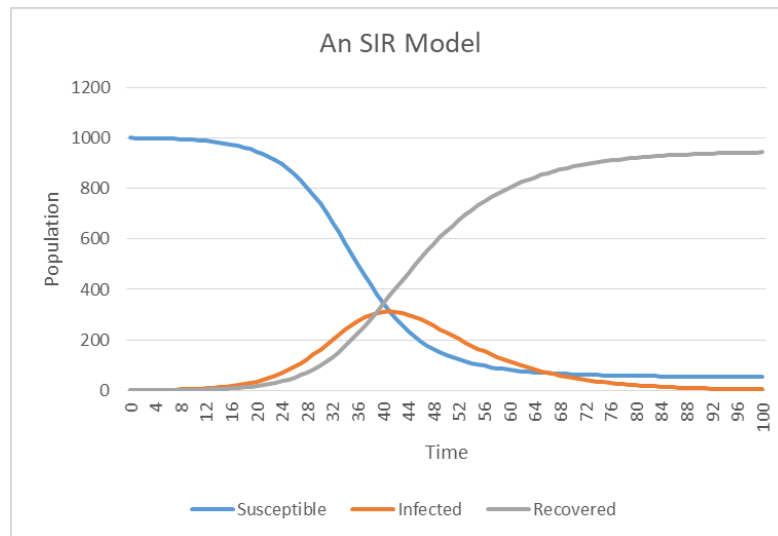
**Susceptible**  $B6 = B5 - B5 * C5 * \text{interaction\_rate} * \text{infection\_rate} * \text{delta\_time}$

**Infected**  $C6 = C5 + B5 * C5 * \text{interaction\_rate} * \text{infection\_rate} * \text{delta\_time} - C5 * \text{recovery\_rate} * \text{delta\_time}$

**Recovered**  $D6 = D5 + C5 * \text{recovery\_rate} * \text{delta\_time}$

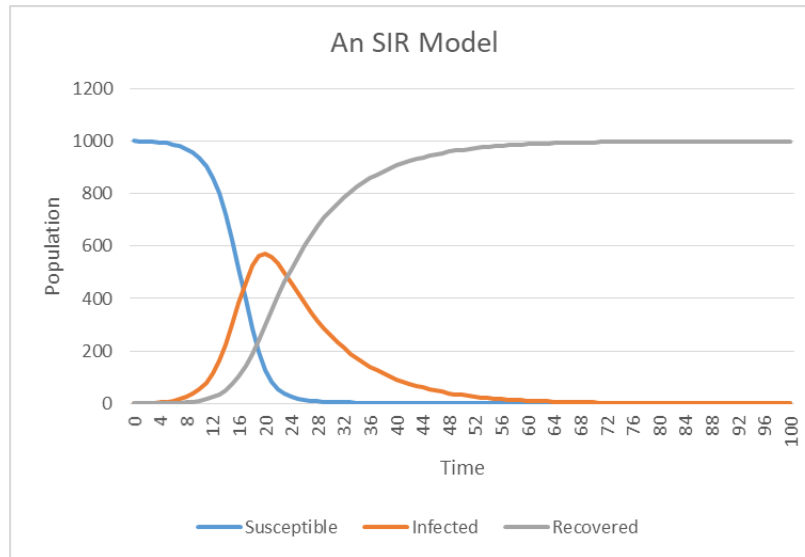
6. Drag all the cells till time = 100.

7. Plot a S,I,R in same graph.



Based on the output, we can tell that as time increases from  $t = 0$  to  $t = 40$  (day 40), the susceptible population gets more infected, thereby increasing the infection graph up until at some point when some population starts getting recovered from the infection, as we note a decrease in both susceptible and infected population. However, at such recovery period, the recovery curve starts rising steadily up until it flattens.

8. Assess the sensitivity of variation in parameters based on the following initial output. For example, maintaining all other parameters, if we change infection rate to 0.4, i.e. Cell I5=40, we observe that the susceptible population curve drops down quickly, as justified by a sharp rise of infected population curve (see output below).



### Solution (b) solving an SIR model using ODE45 method

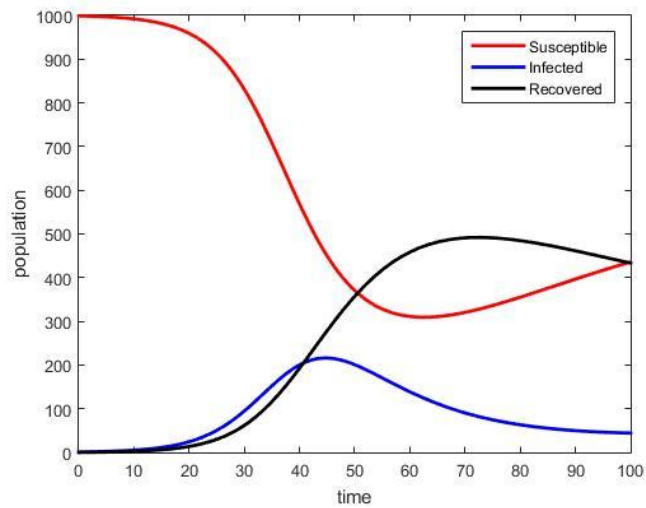
Using the following set of parameters and an Euler ode45 algorithm, we get the following codes in MATLAB

Table 2: Common SIR Model parameters

parameters	values
$N$	1000
$\mu$	1/60
$\beta_1$	20/100
$\beta_2$	15/10000
$\delta$	0.03
$\gamma$	0.1
$I_0$	1
$R_0$	0
$S_0$	$N - I_0 - R_0$

```
function EulerSIR()
clear all; clc;
N=1000;mu=1/60;
beta1=20/100;
beta2=15/10000;sigma=0.02;
gamma=0.1;
options = odeset('RelTol',1e-4,'AbsTol',[1e-4
1e-4 1e-4]);
[T,Y] = ode45(@SIRmodel,[0 100],[N-1 1
0],options);
plot(T,Y(:,1),'r',T,Y(:,2),'b',T,Y(:,3),'k','Lin
ewidth',2);
xlabel('time');
ylabel('population')
legend('Susceptible','Infected','Recovered')

function dy=SIRmodel(~,y)
dy=zeros(3,1);
dy(1)=mu*N-beta1*beta2*y(1)*y(2)-mu*y(1);
dy(2)=beta1*beta2*y(1)*y(2)-
(mu+sigma+gamma)*y(2);
dy(3)= gamma*y(2)-mu*y(3);
end
end
```



### Activity 5 a

- a) Write a MATLAB code that simulates disease dynamics of the SIR model with initial values  $S(0) = 997$ ,  $I(0) = 3$ ,  $R(0) = 0$ , and rates for infection  $\beta = 0.4$  and for recovery  $\gamma = 0.04$ .
  - a. Comment on the graphical outputs.
  - b. Describe the effect of reducing the rate of infection from  $\beta = 0.5$  to  $\beta = 0.12$  [**Hint:** If the disease has no medicine or vaccination available, the only possible mechanism to reduce the infection rate (“flattening the curve”) is by appropriate measures such as social distancing].
- b) Consider the following set of equations describing an SIR model

$$\begin{aligned}\frac{dS}{dt} &= \mu N - \beta SI - \mu S \\ \frac{dI}{dt} &= \beta SI - (\mu + \delta + \gamma)I \\ \frac{dR}{dt} &= \gamma I - \mu R\end{aligned}$$

$$S(0) = S_0, \quad I(0) = I_0, \quad R(0) = R_0$$

where the given variables and parameters have the same meaning as used before. Implement the Euler method to solve for the variables  $S$ ,  $I$  and  $R$ , and parameters appearing in Table 11.





## Ecological modeling

Ecological modeling describes mathematical models and systems for describing ecological processes such as interaction between humans or animals and environment. Simply put, ecological modeling is the construction and analysis of mathematical models of ecological processes both purely biological and combined with biophysical models. These models can be analytic or simulation-based and are used to understand complex ecological processes and predict how real ecosystems might change. In this section, we are going to look at ecological models that have analytical solutions which can then be simulated via MATLAB to depict real life scenario. Three types of ecological models exist subject to their relation with change; temporal (time), spatial (space), and spatial-dynamic (varying space).

### **Example:** Exponential Growth – Population

Let  $P(t)$  be a quantity that increases with time  $t$  and the rate of increase is proportional to the same quantity  $P$  as follows

$$dP / d t = k P$$

where  $d p / d t$  is the first derivative of  $P$ ,  $k > 0$  and  $t$  is the time.  
The solution to the above first order differential equation is given by

$$P(t) = A e^{kt}$$

where  $A$  is a constant not equal to 0.

If  $P = P_0$  at  $t = 0$ , then

$$P_0 = A e^0$$

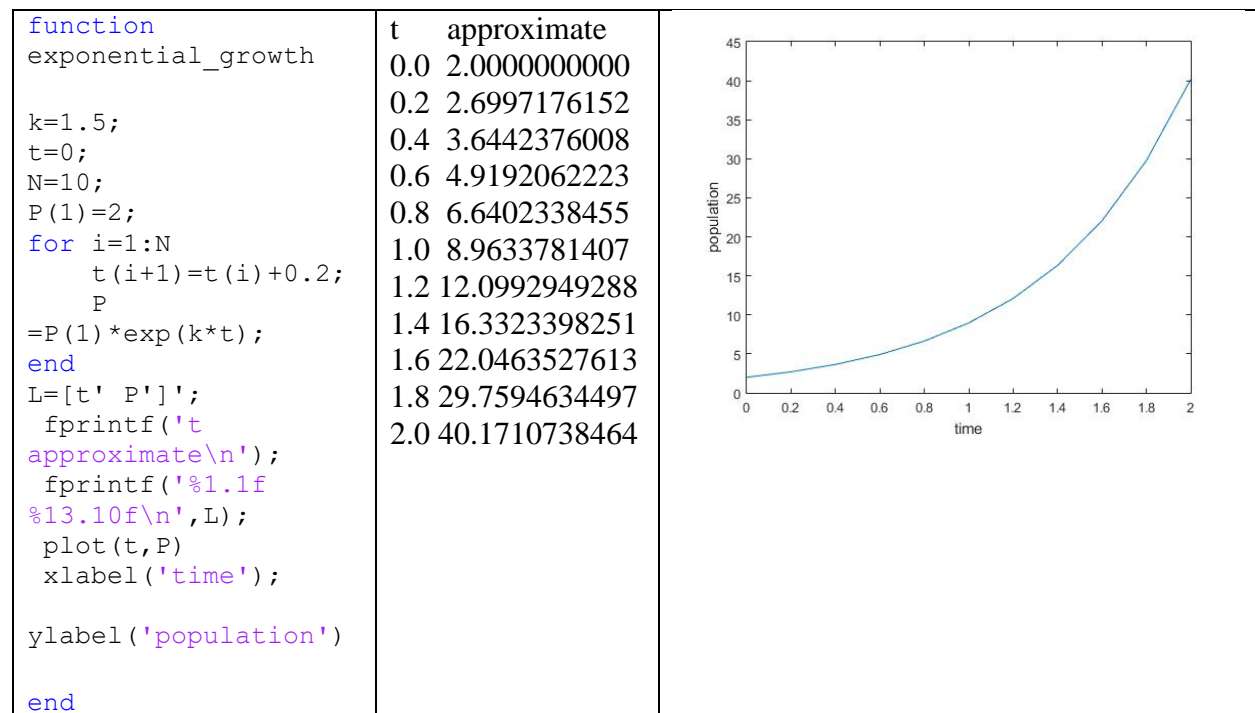
which gives  $A = P_0$

The final form of the solution is given by

$$P(t) = P_0 e^{kt}$$

Assuming  $P_0$  is positive and since  $k$  is positive,  $P(t)$  is an increasing exponential.  $\frac{dP}{dt} = kP$  is also called an exponential growth model.

In MATLAB, we can simulate the solution as



### Example: The Lotka–Volterra equations

One of the earliest and most well-known, ecological models is the predator-prey model of Alfred J. Lotka (1925) and Vito Volterra (1926). This model takes the form of a pair of ordinary differential equations (ODEs), one representing a prey species, the other its predator as follows:

$$\begin{aligned}\frac{dX}{dt} &= \alpha X - \beta XY \\ \frac{dY}{dt} &= \gamma \beta XY - \delta Y\end{aligned}$$

Where

- $X$  – number/concentration of prey species
- $Y$  – number/concentration of predator species
- $\alpha$  – prey species' growth rate
- $\delta$  – predator's mortality rate
- $\beta$  – predation rate of  $Y$  upon  $X$
- $\gamma$  – assimilation efficiency of  $Y$

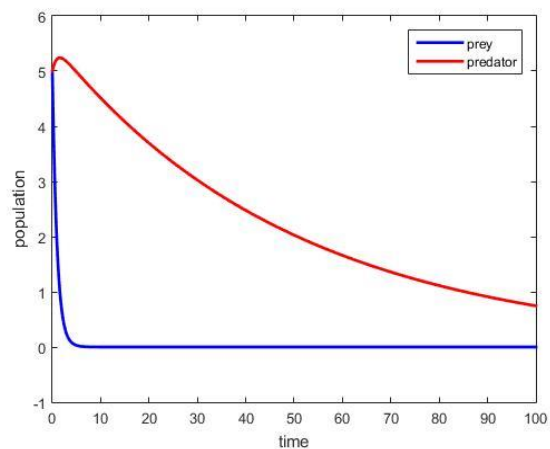
Assuming that Lotka-Volterra system of equations was originally devised to explain fluctuations in fish and shark populations observed in the Adriatic Sea, write a MATLAB function that demonstrates a sample time-series of the model and comment based on the output.

### Solution:

The Lotka-Volterra system of equations has analytical solution using eigenvalue method. However, simulating the same using MATLAB, we have

```
function Lotka_Volterra()
clear all; clc;
beta=20/100;
alpha=15/10000;sigma=0.02;
gamma=0.1;
options = odeset('RelTol',1e-4, 'AbsTol',[1e-4 1e-4]);
[T,Y] = ode45(@Lotkamodel,[0 100],[5 5],options);
plot(T,Y(:,1),'b',T,Y(:,2),'r','Linewidth',2);
xlabel('time');
ylabel('population')
legend('prey','predator')

function dy=Lotkamodel(~,y)
dy=zeros(2,1);
dy(1)=alpha*y(1)-beta*y(1)*y(2);
dy(2)=gamma*beta*y(1)*y(2)-sigma*y(2);
end
end
```



Based on the output, using assumed parameters for  $X = 5, Y = 5, \beta = 0.2, \delta = 0.02$ , and  $\gamma = 0.0015$ , we note that the two populations exhibit parasitic symbiotic behavior such that the predator (shark) curve lags behind that of the prey (fish). As the prey (fish) population gets depleted, the predator (shark) population suffers the consequence as well, by slowly dropping down its population.



## Activity 5 b

- a) Consider a closed biological system populated by  $M$  number of prey and  $N$  number of predators. Volterra postulated that the two populations are related by the set of ordinary differential equations

$$\begin{aligned}\frac{dM}{dt} &= aM - bMN \\ \frac{dN}{dt} &= -cN + dMN\end{aligned}$$

where  $a, b, c$  and  $d$  are constants. The steady state solutions are  $M_0 = c/d$  and  $N_0 = a/b$ ; if numbers other than these are introduced into the system, the populations undergo periodic fluctuations. Introducing the new notations  $y_1 = \frac{M}{M_0}$  and  $y_2 = \frac{N}{N_0}$ , we have a new set of differential equations described as

$$\begin{aligned}\frac{dy_1}{dt} &= a(y_1 - y_1y_2) \\ \frac{dy_2}{dt} &= b(-y_2 + y_1y_2)\end{aligned}$$

Using  $a = 1.0/\text{year}$ ,  $b = 0.2/\text{year}$ ,  $y_1(0) = 0.1$  and  $y_2(0) = 1.0$ , plot the two population graphs from  $t = 0$  to 50 years.

- b) In chemistry or biochemistry, a nonlinear equation  $y = \frac{ax}{x+b}$  is widely used for different research purpose such that Langmuir equation which relates coverage or adsorption of molecules on a solid surface to gas pressure or concentration of a medium above solid surface at fixed temperature is described by

$$\theta = \frac{\alpha P}{1 + \alpha P}$$

where  $\theta$  is the fractional coverage of the surface and  $P$  is gas pressure or concentration in the case of liquids. Plot the graph of the Langmuir equation assuming  $\alpha = 0.02$  (constant) for a finite length of time beginning with  $t = 0$ .

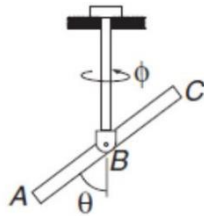


## Science and Engineering

This section was included in this module specifically for students interested in the computational mathematics skills and knowledge required to develop efficient solutions to general industrial problems based on modern programming methods for science and engineering, with an emphasis on numerical methods and algorithms.

### Example

The bar  $ABC$  is attached to the vertical rod with a horizontal pin. The assembly is free to rotate about the axis of the rod.



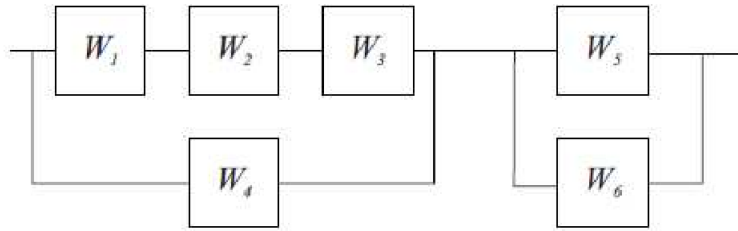
In the absence of friction, the equations of motion of the system are

$$\begin{aligned}\ddot{\theta} &= \dot{\phi}^2 \sin \theta \cos \theta \\ \ddot{\phi} &= -2\dot{\theta}\dot{\phi} \cot \theta\end{aligned}$$

If the system is set into motion with the initial conditions  $\theta(0) = \pi/12 \text{ rad}$ ,  $\dot{\theta}(0) = 0$ ,  $\phi(0) = 0$  and  $\dot{\phi}(0) = 20 \text{ rad/s}$ , obtain a numerical solution with the adaptive Runge-Kutta method from  $t = 0$  to  $1.5 \text{ sec}$  and plot  $\dot{\phi}$  versus  $t$ .

### Example: System Liability

Given a 6 component system of the configuration shown in figure below, use MATLAB simulations to test for the reliability of the system.



### Solution

To simulate 100 trials of the six-component test (such that each component works with a specified probability  $q$ ), we use the following MATLAB function:

```
function N=reliable(n,q)
% n is the number of 6 component devices
% N is the number of working devices
W=rand(n,6)>q;
D=(W(:,1)&W(:,2)&W(:,3))|W(:,4);
D=D&(W(:,5)&W(:,6));
N=sum(D);
end
```

The  $n \times 6$  matrix  $W$  is a logical matrix such that  $W(i, j) = 1$  if component  $j$  of device  $i$  works properly. Note that  $D(i) = 1$  if device  $i$  works. Otherwise,  $D(i) = 0$ . Hence,  $N$  is the number of working devices. Since we are randomly generating the simulations, one possible result of 10 repetitions of the 100 trials for  $q = 0.2$  can be

```
>> for n=1:10
w(n)=reliable(100,0.2);
end
>> w
w =
    55    54    59    61    51    52    60    60    54    57
```

### Example: Exponential Decay - Radioactive Material

Let  $M(t)$  be the amount of a product that decreases with time  $t$  and the rate of decrease is proportional to the amount  $M$  as follows

$$\frac{dM}{dt} = -kM$$

where  $\frac{dM}{dt}$  is the first derivative of  $M$ ,  $k > 0$  and  $t$  is the time. Using method of separation of variables (SOV), we can solve the above first order differential equation to obtain

$$M(t) = A e^{-kt}$$

where  $A$  is non zero constant. If we assume that  $M = M_0$  at  $t = 0$ , then

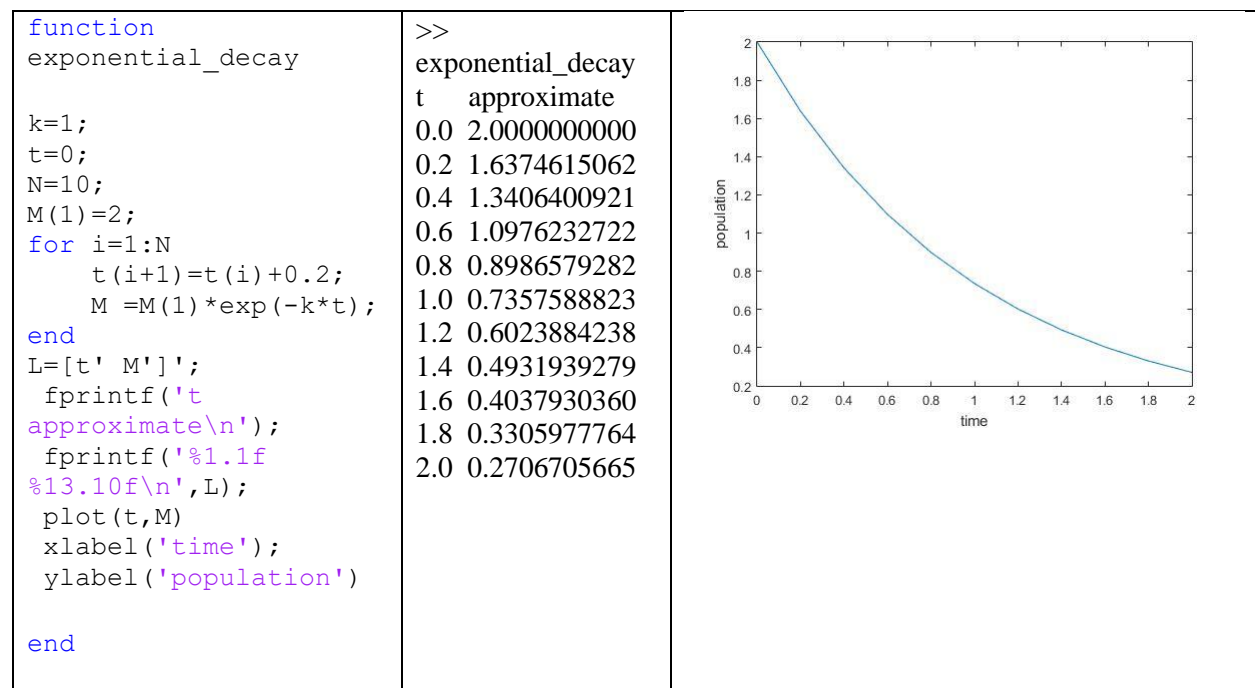
$$M_0 = A e^0$$

which gives  $A = M_0$ . Thus, the solution may be written as follows

$$M(t) = M_0 e^{-kt}$$

Assuming  $M_0$  is positive and since  $k$  is positive,  $M(t)$  is an decreasing exponential.  $\frac{dM}{dt} = -kM$  is also called an exponential decay model.

In MATLAB, we can simulate the solution as





## Activity 5 c

- a) **Falling Object:** An object is dropped from a height at time  $t = 0$ . If  $h(t)$  is the height of the object at time  $t$ ,  $a(t)$  the acceleration and  $v(t)$  the velocity. Given that the relationships between  $a$ ,  $v$  and  $h$  are as follows:

$$a(t) = dv/dt, v(t) = dh/dt.$$

For a falling object,  $a(t)$  is constant and is equal to  $g = -9.8 \text{ m/s}^2$ . Find an expression for  $h(t)$  and simulate the solution in MATLAB.

Combining the above differential equations, we can easily deduce the following equation  
 $d^2h/dt^2 = g$

Integrate both sides of the above equation to obtain

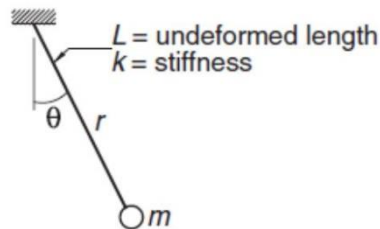
$$dh/dt = gt + v_0$$

Integrate one more time to obtain

$$h(t) = (1/2)gt^2 + v_0t + h_0$$

The above equation describes the height of a falling object, from an initial height  $h_0$  at an initial velocity  $v_0$ , as a function of time.

- b) **Pendulum:** The mass  $m$  is suspended from an elastic cord with an extensional stiffness  $k$  and undeformed length  $L$ .



If the mass is released from the rest at  $\theta = 60^\circ$  with the cord unstretched, find the length  $r$  of the cord when the position  $\theta = 0$  is reached for the first time. The differential equations describing the motion are

$$\frac{d^2r}{dt^2} = r \frac{d^2\theta}{dt^2} + g \cos \theta - \frac{k}{m}(r - L)$$

$$\frac{d^2\theta}{dt^2} = \frac{-2 \frac{dr}{dt} \frac{d\theta}{dt} - g \sin \theta}{r}$$

Use the values  $g = 9.80665 \text{ m/s}^2$ ,  $k = 40 \text{ N/m}$ ,  $L = 0.5 \text{ m}$  and  $m = 0.25 \text{ kg}$  when simulating the solution in MATLAB.





## Statistics

Computational skills, which are often required to solve real-world problems, will be extended for statistical problems in this section. The partnership of applied mathematics, computational mathematics, and statistics brings the tools of modeling, simulation, and data analysis to bear on real-world problems, producing solutions with the power to predict and explain complex phenomena. By the end of this section, we shall observe that these methods, often applied computationally, are being used in a wide variety of areas in business, engineering, the natural sciences, and the social sciences.

### Random Numbers and Simulations

We'll observe that we can use Matlab to generate **random numbers** so that we can discuss its **statistical capabilities** and explore **some simulations**, too. We'll start with a definition of some statistical quantities though.

But first, let us consider a set of data which we'll refer to as  $x_i$  where  $i = 1$  to  $N$ . We'll not make any assumptions about the data in terms of its source or its form.

### Averages

Recall that the mean is given by adding up all the data and dividing by the number of objects in the set. This is written as:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

In fact, there's a Matlab command which does exactly this: '**mean(x)**' and there's another related command, which is '**median(x)**' and it's the value in the 'middle' of a vector when the data has been **sorted**. If there's an odd number of pieces of data this is well defined, however if the number is even then the mean of the data at each end of the central interval is given, for example the median of [1 2 3 4] is  $(2+3)/2 = 2.5$  whereas the median of [1 3 5] is 3. Note that the median can equal the mean but it is not necessarily so.

For example, in Matlab, this short code

```
x = [4 0 5 3 2 1 3 5 9 3 7 5 6 8 2 0];
mn = mean(x)
md = median(x)
```

results in

```
mn = 3.9375
md = 3.5000
```

## Other Statistical Measures

In the just ended section we have discussed averages of a set of data but there are more measures available, for instance these two distributions have the same mean but are obviously different in form:

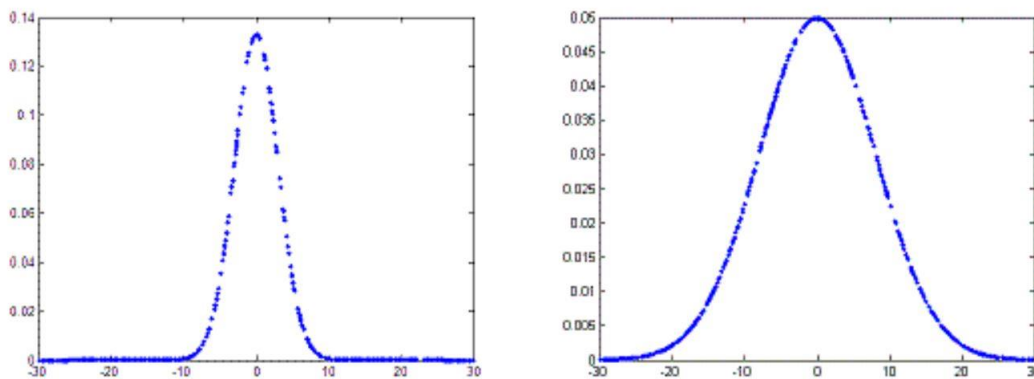


Figure 3: Two different Normal distributions of same mean but different shapes

The first curve is narrower than the second and this is quantified using the variance, which is given by

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$$

We can interpret this value as the ‘**mean**’ of the sum of the squares of the distance from the mean. Of course we are lucky that there’s also a Matlab command ‘**var(x)**’ to calculate this number. Another related measure is the **standard deviation**, which is the square root of the variance, ‘**std(x)**’. This also is a measure of the width of the distribution and has the advantage that it has the same units as the data.

There are other measures, some of which are called ‘**higher-order moments**’ (**mean** is first and **variance** is second): the **skewness** is the third, and the **kurtosis** is the fourth.

## Random Numbers and Distributions

In order to generate random numbers, there are various commands available in Matlab that we can use. We won't worry about how this is done or the technical aspects of the seeding. We're going to start with the simple command '**rand**' which returns a random number between zero and one.

### Pseudo-random number generator

With statistics in general, a pseudo-random number generator produces a sequence of random numbers between 0 and 1, i.e.  $\text{rand}(m, n)$  produces an  $m \times n$  array of uniformly distributed pseudo-random numbers. We shall see that this generator can be utilized in MATLAB when simulating experiments.

Thus, for a MATLAB simulation, we first generate a vector R of N random numbers:

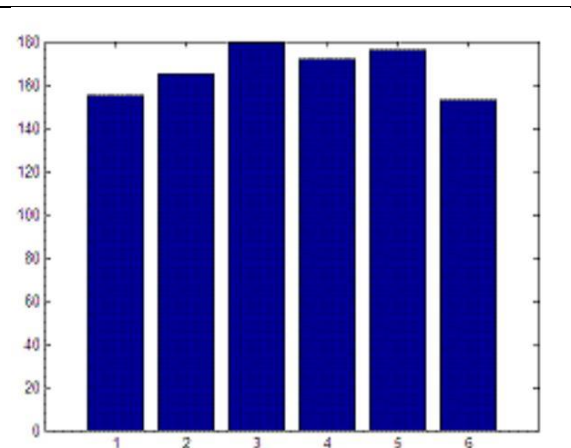
```
>> N=100;  
>> R=rand(1,N);
```

To simulate an experiment that contains an event with probability  $p$ , each random number  $r$  produced by above command will be tested such that if  $r < p$ , event occurs; otherwise it does not occur.

### Example: Rolling a Die

Alternatively, if we rolled a die a large number of times, and we used the above formulas to calculate the mean and variance, we would expect to obtain a *mean* = 3.5 and a *variance* = 2.916. Let's try it with Matlab using a simple dice program.

```
function n = roll_d()  
n = ceil(rand*6 + eps);  
  
% Roll the die and keep the values in d  
for i = 1 : 1000  
    d(i) = roll_d;  
end  
  
% Find how many appearances of each possible value  
for i = 1 : 6  
    v(i) = length(find(d==i));  
end
```

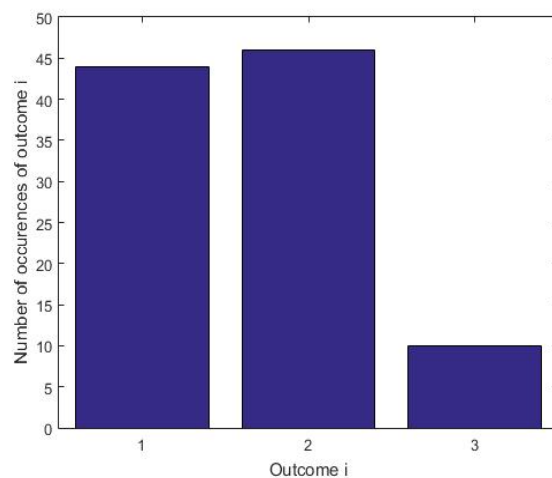


<pre>% Display results disp(['Mean = ' num2str(mean(d))]) disp(['Variance = ' num2str(var(d))]) bar(v)</pre>	<pre>Mean = 3.508 Variance = 2.7827</pre>
--	---

### Example: Tossing a Coin

Suppose we wish to simulate a coin flipping experiment which yields heads with probability 0.4, tails with probability 0.5, or lands on its edge with probability 0.1, and that 100 simulations are needed. In this case, we generate vector  $X$  as a function of  $R$  to represent 3 possible outcomes:  $X(i) = 1$  if flip  $i$  was heads,  $X(i) = 2$  if flip  $i$  was tails, and  $X(i) = 3$  if flip  $i$  landed on the edge. In MATLAB, we implement this experiment as

```
>> N=100;
>> R=rand(1,N);
>>
X=(R<=0.4)+(2*(R>0.4).*(R<=0.9))+(3*(R>0.9));
>> [N,Y]=hist(X,1:3);
>> figure,bar(Y,N)
>> xlabel('Outcome i'),ylabel('Number of occurrences of outcome i')
```



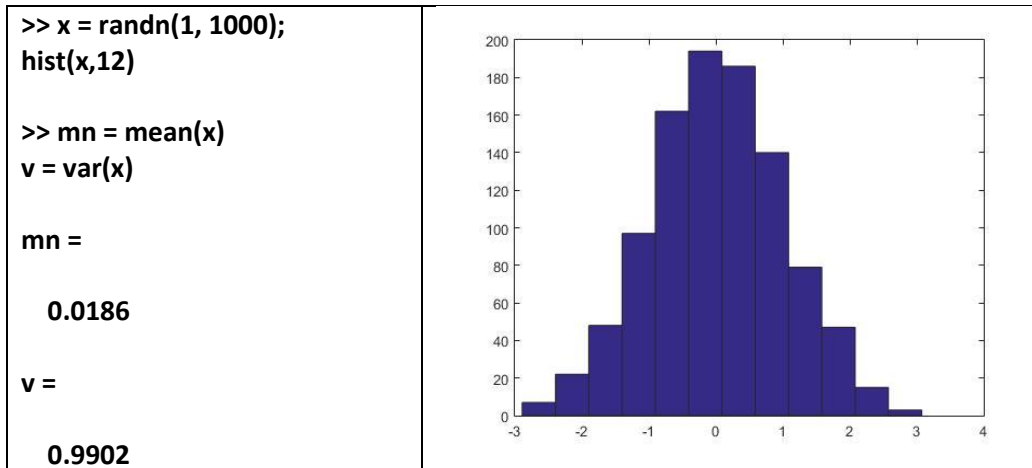
Alternatively

```
function r = toss()
p = ['heads'; 'tails'];
i = ceil(rand*2 + eps);
r = p(i,:);
```

This can then be called just using 'toss' which generates either 'heads' or 'tails'.

### Normal Distribution

A normal distribution has two parameters associated with it: the mean and the variance. The command 'randn' generates random numbers which have a **mean** of zero and a **variance** of unity.



### Example: Flipping Two Coins

The following MATLAB function can be used to simulate the result of a two-coin experiment where this function generates vectors  $C$  and  $H$  for  $n$  trials of this experiment.  $C(i)$  indicates which coin (biased =1 or fair =2) is chosen, and  $H(i)$  is the simulated result of a coin flip with heads,  $H(i) = 1$  occurring with probability  $P(i)$ .

```
function [C, H] = twocoin(n) % n: number of trials
C = ceil(2*rand(n, 1)); % C(i) = 1 if coin 1, C(i) = 2 if coin 2 is
chosen for trial i
P = 1 - (C/4); % P(i) = 0.75 if C(i) = 1, otherwise if C(i) = 2
H = (rand(n, 1) < P);
end
```

For  $n = 5$  number of trials, we get the following output on the command window

```
>> [C, H] = twocoin(5)
C =
     1
     2
     1
     1
     2
H =
     0
     1
     1
     1
     1
```

In statistics, distribution functions play a major role in describing the nature of experiments, as in either Geometric, Hypergeometric, Binomial, Poisson, Exponential, Uniform, etc. what is key therefore is to know how to derive the probability distributions functions, called probability mass functions (PMF) or probability density function (PDF) for the discrete or continuous random variables, respectively. Beyond PMF and PDF we can compute cumulative probabilities, respectively, called cumulative distribution function (CDF).

For discrete random variables, the PMFs and CDFs for the families of random variables can be computed. For example, for a finite discrete random variable  $X$  defined by a set of sample values  $S_x = \{s_1, s_2, \dots, s_n\}$  with corresponding probabilities  $p_i = P_X(s_i) = P[X = s_i]$ ,  $p = [p_1, p_2, \dots, p_n]$ , the following MATLAB function returns the PMF  $y_i = P_X(x_i)$ .

<pre>function pmf=finitepmf(sx,px,x) % finite random variable X: % vector sx of sample space % elements {sx(1),sx(2), ...} % vector px of probabilities % px(i)=P[X=sx(i)] % Output is the vector % pmf: pmf(i)=P[X=x(i)] pmf=zeros(size(x(:))); for i=1:length(x)     pmf(i)= sum(px(find(sx==x(i)))); end</pre>	<pre>&gt;&gt; sx=[1 2 3 6 5 6]; % 6 outcomes of a fair die &gt;&gt; px=[1/6 1/6 1/6 1/6 1/6 1/6]; % equal chances &gt;&gt; x=2; % if a die turns out a face of 2 after toss &gt;&gt; pmf=finitepmf(sx,px,x)</pre> <p>pmf =</p> <p>0.1667</p>
---	--

Through the same definition, the CDF can simply be found by

<pre>function cdf=finitecdf(sx,px,x) % finite random variable X: % vector sx of sample space % elements {sx(1),sx(2), ...} % vector px of probabilities % px(i)=P[X=sx(i)] % Output is the vector % cdf: cdf(i)=P[X=x(i)] cdf=[]; for i=1:length(x)     pxi= sum(px(find(sx&lt;=x(i))));     cdf=[cdf; pxi]; end</pre>	<pre>&gt;&gt; sx=[1 2 3 6 5 6]; &gt;&gt; px=[1/6 1/6 1/6 1/6 1/6 1/6]; &gt;&gt; x=5; &gt;&gt; cdf1=finitecdf(sx,px,x)</pre> <p>cdf1 =</p> <p>0.6667</p>
--	---

Note that the CDF can also be computed by summing the PMF as

```
>> cdf=cumsum(pmf);
```

### Example: CDF of a Binomial Distribution

The sample values of random variables can be generated based on the calculation of the CDF first. For example, the  $m$  samples of a  $\text{binomial}(n, p)$  random variable can be generated as

```
function x=binomialrv(n,p,m)
% m binomial(n,p) samples
r=rand(m,1);
cdf=binomialcdf(n,p,0:n);
x=count(cdf,r);
end
```

#### Exercise

Define a binomial CDF and complete running this code

### Example: Permutation & Combination

Recall from Discrete Mathematics module or College Algebra module that **permutation** of a number of objects is the number of different ways they can be ordered: the position is important. However, with **combinations**, one does not consider the order in which objects were placed.

Study the following MATLAB Code and use it to answer the three questions that follow;

```
% Clears variables and screen
clear; clc
% Asks user for input
n = input('Total number of objects: ');
d = input('Size of subgroup: ');
% Computes and displays permut. according to basic formulas
p = 1;
for i = n - d + 1 : n
    p = p*i;
end
str1 = [num2str(p) ' permutations'];
disp(str1)
% Computes and displays combin. according to basic formulas
str2 = [num2str(p/factorial(d)) ' combinations'];
disp(str2)
```

- (a) In not more than two sentences, what does the code do?
  - a. This algorithm (program in Matlab) calculates the number of permutations and combinations of  $N$  objects taken  $D$  at a time.
- (b) How many permutations and combinations can be made of the 26 letters of the alphabet, taking five at a time?
  - a. We run the code above and enter: Total number of objects: 26 and Size of subgroup: 5. The answer is: 7893600 permutations and 65780 combinations

- (c) How many different ways can 12 computers be repaired if the workshop can only support 2 at a time?
- We run the Matlab m-file above and enter: Total number of objects: 12 and Size of subgroup: 2. The answer (with no doubt) is: 132 permutations and 66 combinations

### Example: T-statistic, Student's t-distribution Test

In this example, we describe a program that performs the t-statistic and degrees of freedom for Student's distribution, in Matlab code. The calculations will be based on any one of these three hypotheses.

#### Hypothesis 1:

The first hypothesis assumes that one population mean is equal to a given value. We shall have to enter the elements of the sample and the value of the mean.

#### First case: one sample given, one mean assumed

$$temp = \frac{\left( d - \frac{v^2}{r} \right)}{r - 1}$$

$$t = (m - gm) \sqrt{\frac{r}{temp}}$$

$$df = r - 1$$

where:

$t$  = t-statistic result

$df$  = degrees of freedom

$v$  = sum of elements

$d$  = sum of squared elements

$r$  = number of elements in sample

$m$  = mean of current sample

$gm$  = assumed mean

This is how we can implement the formulas in Matlab:

```
function [t, df] = t_test1(gv, gm)
% Inputs: gv = given vector
%          gm = assumed mean
% Outputs: t = t-test
%          df = degrees of freedom

% number of elements in vector
r = length(gv);
```



```

% sum of elements
v = sum(gv);
% sum of squared elements
d = sum(gv.^2);
% mean of current vector
m = mean(gv);

% calculation of t-test and degrees of freedom
% Case 1: one vector with assumed mean
av = (d - v^2 / r) / (r - 1);
t = (m - gm) * sqrt(r/av);
df = r - 1;

```

## Hypothesis 2:

The other hypotheses compare two samples. In both tests the means of the two populations are equal, but the standard deviations may be equal or different. For these hypotheses we shall enter the elements of each sample.

### Second case: two samples given with equal mean and std. deviation

$$temp_1 = \frac{\left(d_1 - \frac{v_1^2}{r_1}\right)}{r_1 - 1} \quad temp_2 = \frac{\left(d_2 - \frac{v_2^2}{r_2}\right)}{r_2 - 1}$$

$$temp_3 = \frac{(m_1 - m_2)}{\sqrt{\frac{1}{r_1} + \frac{1}{r_2}}}$$

$$df = r_1 + r_2 - 2$$

$$t = \frac{temp_3}{\sqrt{\frac{(r_1 - 1)temp_1 + (r_2 - 1)temp_2}{df}}}$$

where:

$t$  = t-statistic result

$df$  = degrees of freedom

$v_{1,2}$  = sum of elements in samples 1 and 2

$d_{1,2}$  = sum of squared elements in samples 1 and 2

$r_{1,2}$  = number of elements in samples 1 and 2

$m_{1,2}$  = mean of samples 1 and 2

This is how we can implement the second case in Matlab:

```
function [t, df] = t_test2(gv1, gv2)
% Inputs: gv1 = given sample 1
%          gv2 = given sample 2
% Outputs: t = t-test
%          df = degrees of freedom

% number of elements in vector 1
r1 = length(gv1);
% sum of elements in vector 1
v1 = sum(gv1);
% sum of squared elements in vector 1
d1 = sum(gv1.^2);
% mean of vector 1
m1 = mean(gv1);

% number of elements in vector 2
r2 = length(gv2);
% sum of elements in vector 2
v2 = sum(gv2);
% sum of squared elements in vector 2
d2 = sum(gv2.^2);
% mean of vector 2
m2 = mean(gv2);

% calculation of t-test and degrees of freedom
% Case 2: two samples, equal mean and std. dev.
av1 = (d1 - v1^2 / r1) / (r1 - 1);
av2 = (d2 - v2^2 / r2) / (r2 - 1);
av3 = (m1 - m2) / sqrt(1/r1 + 1/r2);
df = r1 + r2 - 2;
t = av3 / sqrt(((r1-1)*av1 + (r2-1)*av2) / df);
```

**Third case: two samples given with equal mean but different std. deviation**

$$temp_1 = \frac{\left(d_1 - \frac{v_1^2}{r_1}\right)}{r_1 - 1} \quad temp_2 = \frac{\left(d_2 - \frac{v_2^2}{r_2}\right)}{r_2 - 1}$$

$$temp_3 = \frac{temp_1}{r_1} + \frac{temp_2}{r_2}$$

$$t = \frac{m_1 - m_2}{\sqrt{temp_3}}$$

$$df = \frac{temp_3^2}{\frac{\left(\frac{temp_1}{r_1}\right)^2}{r_1 + 1} + \frac{\left(\frac{temp_2}{r_2}\right)^2}{r_2 + 1}} - 2$$

again:

$t$  = t-test result

$df$  = degrees of freedom

$v_{1,2}$  = sum of elements in samples 1 and 2

$d_{1,2}$  = sum of squared elements in samples 1 and 2

$r_{1,2}$  = number of elements in samples 1 and 2

$m_{1,2}$  = mean of samples 1 and 2

This is our third case in Matlab code:

```
function [t, df] = t_test3(gv1, gv2)
% Inputs: gv = given vector
%          gm = assumed mean
% Outputs: t = t-test
%          df = degrees of freedom

r1 = length(gv1);
v1 = sum(gv1);
d1 = sum(gv1.^2);
m1 = mean(gv1);

r2 = length(gv2);
v2 = sum(gv2);
d2 = sum(gv2.^2);
m2 = mean(gv2);

% calculation of t-test and degrees of freedom
% Case 3: two samples, equal mean but different std. dev.
```

```

av1 = (d1 - v1^2 / r1) / (r1 - 1);
av2 = (d2 - v2^2 / r2) / (r2 - 1);
av3 = av1/r1 + av2/r2;
t = (m1 - m2) / sqrt(av3);
df = round(av3^2/((av1/r1)^2/(r1+1) + (av2/r2)^2/(r2+1)) - 2);

```

### Example: Chi-square distribution

In this example, we shall discuss a program which calculates the tail-end and percentile values for points on a **Chi-square ( $X^2$ ) distribution curve**. Needed to provide are the values of  $X^2$  and the degrees of freedom. Just like in other previous examples, there shall be no special instruction or Matlab toolbox to be used. Since the summation in the calculation of  $Z$  cannot actually extend to infinity, we stop summation when the next term is less than a chosen level of precision. Our precision is limited to approx.  $1 \times 10^{-7}$ .

The Matlab program to accomplish this, is:

```

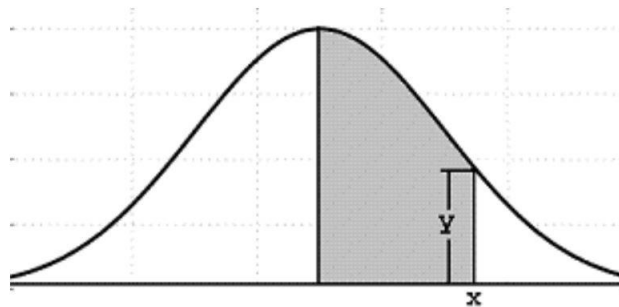
% Clears screen and variables in workspace
clc; clear
% Asks the user for the relevant input
v = input('Degrees of freedom: ');
w = input('Chi-square: ');
% Calculates the denominator product
r = 1;
for i = v : -2 : 2
    r = r*i;
end
% Calculates the numerator product
k = w^(floor((v+1)/2)) * exp(-w/2)/r;
% If degrees of freedom are odd, then uses the pi factor
if floor(v/2) == v/2
    j = 1;
else
    j = sqrt(2/(w*pi));
end
l = 1;
m = 1;
v = v + 2;
m = m*w/v;
% Summation factor
while m >= 1e-7
    l = l + m;
    v = v + 2;
    m = m*w/v;
end
% Displays results
str = ['Tail end value: ' num2str(1-j*k*l)];
disp(str)
str = ['Percentile: ' num2str(j*k*l)];
disp(str)

```



## Activity 5 d

- a) Normal distribution: Consider the following analytical description of estimating probability using the standard normal distribution ;



### Standard Normal distribution

The shaded area represents the probability of 'x', and its frequency is 'y'.  
The normal probability is approximated using the following formula:

$$\text{probability} = 1 - r(a_1t + a_2t^2 + a_3t^3) + \text{eps}$$

where:

$$a_1 = 0.4361836$$

$$a_2 = -0.1201676$$

$$a_3 = 0.9372980$$

$$r = (e^{-x^2/2}) / \sqrt{2\pi}$$

$$t = 1/(1 + 0.33267x)$$

$$\text{eps} < 10^{-5}$$

Write an algorithm (program in Matlab) that calculates the probability and frequency of given values on a standard normal distribution curve (Gauss' bell). [Hint: You have to enter the mean, the standard deviation and the value of interest. Do not use any special toolboxes or strange instructions].

- b) Use the Matlab code developed in question above to respond to the following questions;
- The mean instructions per millisecond (IPMS) of a certain type of modern computers is 150. The standard deviation is 15 IPMS. If the IPMS are normally distributed, what is the probability that a computer executes between 150 and 180 instructions per millisecond?
  - What if we need to know the probability of an execution of 130 and 150 instructions per milliseconds for the same type of modern computers?

- c) A sample of children's IQs was taken, the result being 101, 99, 120, 79, 111, 98, 106, 112, 87, and 97. Calculate the t-statistic assuming the population mean is 100.

A second sample was taken, with a result of 101, 95, 130, 150, 75, 79, 111, 100, 98, and 91. Calculate the t-statistic based on the hypothesis that the two samples have equal means and standard deviations. Calculate again supposing that the std. deviation is different between the groups.

- d) For the following questions under random number generation
- i. Write a Matlab code that you can use to generate 100 uniform random numbers between 0 and 1.
  - ii. Suppose that we have a vector  $\mathbf{x}$  that has values from -10 to 10 in 0.05-steps. Write a Matlab code which we can use to take three random elements from that table.



## Summary/Let Us Sum Up

To sum up, this unit was meant to introduce the student to the application of mathematical computing techniques to solving different real life application problems through four case study areas; disease modeling, ecological modeling, science and engineering, and statistics. Such techniques, often applied computationally, are being used in a wide variety of areas in business, engineering, the natural sciences, and the social sciences. In particular usage of differential equations and probability simulations, are the useful techniques. Below, the reader finds further reading reference materials (books and journals) not only for this Unit, but also in other units.



## Suggestions for Further Reading/ Reading Assignment

- Holzbecher, E. (2012). *Environmental Modeling Using MATLAB* Springer
- Ecological Modelling-International Journal on Ecological Modelling and Systems Ecology, <https://www.journals.elsevier.com/ecological-modelling>
- Rui DIL~AO. (Juin, 2004). CHAPTER 15: Mathematical models in population dynamics and ecology, Institut des Hautes Etudes Scientifiques, <http://preprints.ihs.fr/2004/M/M-04-26.pdf>
- Wei-Bin Zhang, MATHEMATICAL MODELS IN ECONOMICS – Vol. I - Mathematical Models in Economics, Ritsumeikan Asia Pacific University, Oita-ken 874-8577, Japan <https://www.eolss.net/Sample-Chapters/C02/E6-154-00-00.pdf>
- Jeffries, C. (1989). *Mathematical Modeling in Ecology: A Workbook for Students*. Boston: Birkhäuser Basel. DOI: 10.1007/978-1-4612-4550-6
- Jackson, L.I., Trebitz, A.S., & Cottingham, K.L. (2000). An Introduction to the Practice of Ecological Modeling, *BioScience*, **50** (8): 694-706, [https://doi.org/10.1641/0006-3568\(2000\)050\[0694:AITTPO\]2.0.CO;2](https://doi.org/10.1641/0006-3568(2000)050[0694:AITTPO]2.0.CO;2)
- Schuwirth, N., Borgwardt, F., Domisch, S., Friedrichs, M., et al. (2019). How to make ecological models useful for environmental management, *Ecological Modelling*, **411**, <https://doi.org/10.1016/j.ecolmodel.2019.108784>
- Blum, W., & Borromeo-Ferri, R. (2009). Mathematical Modelling: Can It Be Taught And Learnt? In *Journal of Mathematical Modelling and Application* 1/1, 45-58.

- Gilbert, J. K. (2004) Models and Modelling: Routes to more authentic science education. In International Journal of Science and Mathematics Education 2, 115-130.
- Goldhausen, I., Di Fuccia, D.S. (2014). Mathematical Models in Chemistry Lessons', Proceedings of the International Science Education Conference (ISEC) 2014, 25-27 November 2014, National Institute of Education, Singapore
- Goldhausen, I. (2015). Mathematische Modelle im Chemieunterricht, Berlin: uni-edition
- Ines Schmidt, David-S. Di Fuccia (n.d). Mathematical Models in Chemistry Lessons, <https://conference.pixel-online.net/NPSE/files/npse/ed0003/FP/0099-SSE94-FP-NPSE3.pdf>
- Okino, M.S., & Mavrovouniotis, M.L. (1998). Simplification of Mathematical Models of Chemical Reaction Systems. Chemical Reviews, **98**(2), 391-408. DOI: 10.1021/cr950223l
- Martinez-Luaces, V. (2015). Mathematical models for chemistry and biochemistry service courses. CERME 9 - Ninth Congress of the European Society for Research in Mathematics Education, Charles University in Prague, Faculty of Education; ERME, Feb 2015, Prague, Czech Republic. pp.904-909. <https://hal.archives-ouvertes.fr/hal-01287264/document>
- UKEssays. (Nov, 2018). Mathematical Modeling of Disease Epidemics. Retrieved from <https://www.ukessays.com/essays/sciences/mathematical-modeling-of-disease-epidemics.php?vref=1>
- Alharbi, A. (Oct, 2016). COMARA: Computational Mathematics in Real-life Applications, Special session at ADVCOMP 2016, The Tenth International Conference on Advanced Engineering Computing and Applications in Sciences October 9 - 13, 2016 - Venice, Italy. [https://iaria.org/conferences2016/filesADVCOMP16/COMARA\\_ADVCOMP2016.pdf](https://iaria.org/conferences2016/filesADVCOMP16/COMARA_ADVCOMP2016.pdf)