Process Control with MATLAB/Simulink

Process Control with MATLAB/Simulink:

A Guide for Beginners

By

Francisco Javier Rivas

Cambridge Scholars Publishing

Process Control with MATLAB/Simulink: A Guide for Beginners

By Francisco Javier Rivas

This book first published 2024

Cambridge Scholars Publishing

Lady Stephenson Library, Newcastle upon Tyne, NE6 2PA, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN: 978-1-0364-1535-8
ISBN (Ebook): 978-1-0364-1536-5

TABLE OF CONTENTS

STABILITY OF CLOSED LOOP CONTROL SYSTEMS

CONTROLLER TUNING

STATE SPACE APPROACH

DISCRETE SYSTEMS

PREFACE

MATLAB stands as a cornerstone in the realm of process control teaching, empowering students and educators alike with a comprehensive suite of tools for modeling, analysis, and implementation of control systems. The integration of MATLAB into the curriculum has revolutionized the way process control concepts are taught and understood, offering a dynamic platform for exploration and experimentation.

One of the most notable contributions of MATLAB to process control teaching is its robust support for system modeling and simulation. MATLAB's extensive library of built-in functions and toolboxes offers students access to a wide range of analysis and design techniques for control systems. From classical PID control to advanced model predictive control (MPC) algorithms, MATLAB provides students with the flexibility to explore various control strategies and methodologies, gaining insights into their performance and robustness in different scenarios.

Through interactive simulations and virtual experiments, students can develop an intuitive understanding of control concepts, such as stability, transient response, and frequency domain analysis, enhancing their ability to design and tune control systems effectively.

In the realm of process control, certain academic texts (really very high-quality texts) explore the utilization of MATLAB for tackling fundamental tasks such as defining transfer functions, determining root loci, generating Bode plots, and analyzing open loop or closed loop responses. However, this text ventures beyond mere instruction by not only presenting the necessary commands but also furnishing complete programs that encompass a wide array of control-related activities, including controller tuning, compensator design, state space representation in various canonical forms, etc. Furthermore, it offers guidance on utilizing pertinent applications within MATLAB's Control System Toolbox. Some problems are addressed through both MATLAB code and Simulink® block diagrams, providing comprehensive understanding and application.

Structured to progress in complexity, this book navigates a logical sequence in process control education. While theoretical foundations are briefly

introduced at the outset of each section, this text is not confined to traditional pedagogy; rather, it serves as a dynamic supplement to existing resources. Despite the author's non-specialization in programming, the codes presented are assimilable and potentially advantageous for student comprehension. However, it's acknowledged that there's room for refinement to enhance elegance and efficiency, a prospect that could further augment its utility in educational contexts.

CHAPTER 1

APPROACHING THE TIME PROCESS DYNAMIC

1. Time domain system dynamics

Every industrial process must be controlled with various objectives, including safety, production, economics, and more. The control of such processes involves identifying a set of variables to be regulated and another set of variables that impact the former. The connection between the controlled variables and the control variables is established through a mathematical model. Typically, this model consists of a series of differential equations, which may vary in complexity, and, in most cases, must be solved within the time domain.

The advancement of software programs has significantly streamlined the resolution of differential equations. Nevertheless, classical control theory has predominantly relied on the conversion of nonlinear differential equations into linear differential equations. Consequently, following the guidelines of classical control, the first step to be taken is the linearization of systems represented by nonlinear equations.

1.1. Linearization

A system is classified as linear when it adheres to the principle of superposition. This principle states that the response resulting from the simultaneous application of two distinct input functions is the sum of the individual responses. Consequently, in a linear system, the response to multiple inputs is determined by handling one input at a time and then combining the results (Franklin, Powell and Emami-Naeini, 2019).

The process of linearizing nonlinear systems relies on utilizing the Taylor series expansion (Stephanopoulos, 1984), wherein truncation occurs after the second-order term. Hence, for a time invariant (constant coefficients) differential equation f=(x_1, x_2) of two variables (x_1 and x_2), Taylor expansion around the so called nominal or equilibrium state ($\bar{x}_1$ and $\bar{x}_2$) would be:

$$f(x_1,x_2) \approx f(\bar{x}_1,\bar{x}_2) + \sum_{i=1}^{i=n} \left[\left(\frac{\partial^n f(x_1,x_2)}{\partial {x_1}^n} \right)_{\bar{x}_1,\bar{x}_2} \frac{\left(x_1-\bar{x}_1\right)^n}{n!} + \left(\frac{\partial^n f(x_1,x_2)}{\partial {x_2}^n} \right)_{\bar{x}_1,\bar{x}_2} \frac{\left(x_2-\bar{x}_2\right)^n}{n!} \right] \quad [1.1]$$

Truncation beyond the second term would finally lead to:

$$f(x_1,x_2) \approx f(\bar{x}_1,\bar{x}_2) + \left(\frac{\partial f(x_1,x_2)}{\partial x_1} \right)_{\bar{x}_1,\bar{x}_2} \left(x_1-\bar{x}_1\right) + \left(\frac{\partial f(x_1,x_2)}{\partial x_2} \right)_{\bar{x}_1,\bar{x}_2} \left(x_2-\bar{x}_2\right) \quad [1.2]$$

Nonlinear expressions can be linearized in MATLAB by the command: *taylor(f,[x_1 x_2],[x_{1o} x_{2o}],'Order',2)*. This command approximates f (which in this case depends on x_1 and x_2) with the Taylor series expansion at the nominal point [x_{1o} x_{2o}] truncating from the second term.

Typical examples of nonlinear terms in process modeling can be observed in various phenomena, such as gravity-driven tank drainage, reaction rate expressions, heat transfer, and more.

Example 1.1. Linearize around the nominal value ($\bar{h}=h_0$) the mass balance that accounts for the gravitational drainage of liquid from a tank, described by the following expression:

$$A\frac{dh}{dt} = Q_{in} - Q_{out} = Q_{in} - K\sqrt{h} \quad [1.3]$$

Where Q_{in} and Q_{out} are the inlet and outlet flowrate, A is the transversal area of the tank, K is a constant, and h is the height of the liquid in the tank. In Eq. [1.3], the only nonlinear term is the one containing the square root of h. MATLAB will be executed in symbolic form (command *syms* is used to define symbolic variables).

In the following code, as stated previously, instruction *syms* creates the symbolic variables, F is the symbolic term to be linearized while F_approach determines the *taylor* second order expansion of F.

```
syms h ho K
F=-K*sqrt(h); % Term to be linearized
F_approach=taylor (F,h,ho,'order',2) % Term expanded in Taylor series up to the 2nd order

OUT
F_approach = - K*ho^(1/2) - (K*(h - ho))/(2*ho^(1/2))
```

The result obtained from MATLAB leads to the following transformation of eq. [1.3]:

$$A\frac{dh}{dt}=Q_{in}-Q_{out}\approx Q_{in}-K\sqrt{\bar{h}}-\frac{K}{2\sqrt{\bar{h}}}(h-\bar{h}) \qquad [1.4]$$

At steady state conditions, the derivative term is zero:

$$0=\bar{Q}_{in}-\bar{Q}_{out}=\bar{Q}_{in}-K\sqrt{\bar{h}}\Rightarrow\bar{Q}_{in}=K\sqrt{\bar{h}} \qquad [1.5]$$

And eq. [1.4] can be converted to deviation variables (actual value – steady state value):

$$A\frac{dh}{dt}=Q_{in}-Q_{out}\approx(Q_{in}-\bar{Q}_{in})-\frac{K}{2\sqrt{\bar{h}}}(h-\bar{h})=Q'_{in}-K'\times h' \qquad [1.6]$$

Differences after linearization can be obtained if considering, for example, that an empty tank is first filled with the following variable values (arbitrary units) Q_{in}= 5, A =3, K=1.2. Solving differential equations in Matlab is completed by the command *dsolve (differential equation, integration conditions)*. In the following code, dh/dt is expressed as *diff*(h,t) while the initial condition is defined by h(0). The command *ezplot* allows the representation of the function generated by *dsolve* in the interval specified.

```
syms h(t)
Qin = 5; % Inlet flowrate
K = 1.2;
A=3; % Area
t_simu = 100; % Simulation time
h_eq = (Qin/K)^2; % Equilibrium height
% DIFFERENTIAL EQUATION AND INITIAL CONDITION
DEFINITION
eqn = diff(h, t) ==1/A*( Qin - K*sqrt(h)); % Derivative is coded as diff
cond = h(0) == 0; % initial condition
% SOLUTION OF DIFFERENTIAL EQUATION
hSol(t) = dsolve(eqn, cond); % Differential equations are solved by dsolve
% Linearized equation and initial condition
eqn_lin = diff(h, t) == 1/A*(Qin - K*sqrt(h_eq) - K/2/sqrt(h_eq)*(h -
h_eq));
cond = h(0) == 0;
% Solution of linearized differential equation
hSol_linear(t) = dsolve(eqn_lin, cond);
% Plot both solutions in the same graph
figure; % Creates a new figure
hold on; % Allows for multiple curves in the same figure
% Plot the solution of the nonlinear equation
h_nonlinear = ezplot(hSol(t), [0 t_simu]);
set(h_nonlinear, 'Color', 'blue', 'LineStyle', '-'); % Line properties
axis([0 t_simu 0 20]) % Axes size
% Plot the solution of the linearized equation
h_linear = ezplot(hSol_linear(t), [0 t_simu]);
axis([0 t_simu 0 20])
set(h_linear, 'Color', 'red', 'LineStyle', '--');
hold off; % Terminates multiple curves in the same figure
xlabel('Time');
ylabel('h(t)');
title('Differential equation solution');
legend('Non linear', 'Linearized');
grid on;
```

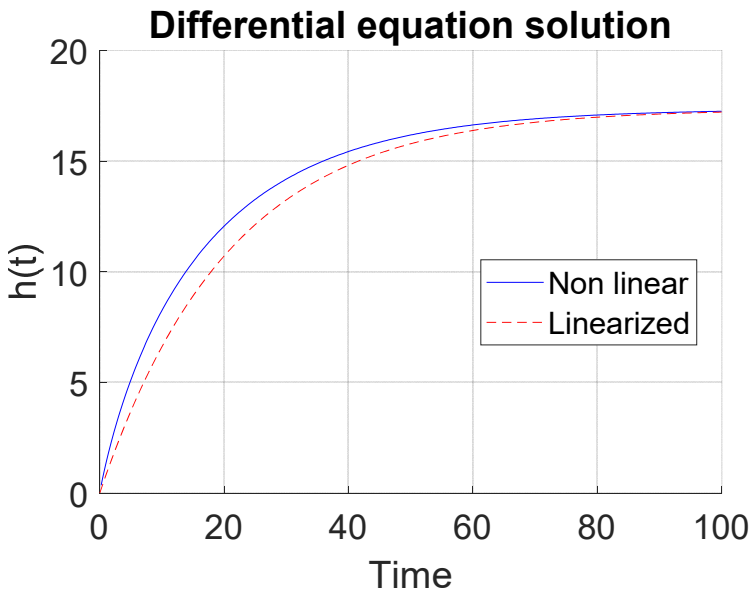


Figure 1.1. Height evolution of liquid in a tank simulated by the nonlinear and linearized differential equations.

Example 1.2. Linearize the energy balance in a perfectly mixed batch reactor at constant volume.

The equation representing the energy balance is given by:

$$A\rho c_p h\frac{dT}{dt}=F_e\rho c_p(T_e-T)+Q \qquad [1.7]$$

Where A is area, ρ density, c_p specific heat, h heigh, T temperature, F_e inlet and outlet flowrate, T_e inlet temperature, and Q stands for the heat exchanged with the surroundings. If F_e is not considered as constant, two nonlinear terms are detected. Notice that the derivative term will be assigned the variable: DT = dT/dt.

The term $h\frac{dT}{dt} = h\times DT$ with steady state conditions $\bar{h}=h_o$ and $DT_o = 0$

```
syms h ho DT DTo
F=h*DT;
F_approach=taylor (F,[h DT],[ho DTo],'order',2)

OUT
F_approach = DTo*ho + ho*(DT - DTo) + DTo*(h - ho)
```

The term $F_e \rho c_p (T_e - T)$

```
syms Fe Fe0  Te Te0  T T0 ro cp
F=Fe*ro*cp*(Te-T);
F_approach=taylor (F,[Fe Te T],[Fe0 Te0 T0],'order',2)
FS=simplify(F_approach)

OUT
F_approach =
Fe0 cp ro (Te - Te0) - Fe0 cp ro (T0 - Te0) - Fe0 cp ro (T - T0) - cp ro (Fe
- Fe0)(T0 - Te0)
FS = -cp*ro*(Fe*T0 + Fe0*T - Fe0*T0 - Fe*Te0 - Fe0*Te + Fe0*Te0)
```

Operating:

$$F_e \rho c_p (T_e - T) \approx c_p \rho Fe(\bar{T}_e - \bar{T}) + c_p \rho \bar{F}e(T_e - T) - c_p \rho \bar{F}e(\bar{T}_e - \bar{T}) \qquad [1.8]$$

Finally, in deviation variables:

$$A\bar{h}\rho c_p \frac{dT'}{dt} = \bar{F}_e \rho c_p (T_e' - T') + F_e' \rho c_p (\bar{T}_e - \bar{T}) + Q' \qquad [1.9]$$

Example 1.3. Consider a non-isothermal CSTR in which a first order reaction occurs. Linearize the component A mass balance.

The equation representing the reagent A mass balance in a CSTR is:

$$V \frac{dC_A}{dt} = F_e (C_{A\_inlet} - C_A) - A_o \exp\left(-\frac{Ea}{RT}\right) C_A \times V \qquad [1.10]$$

Where C_A and C_{A_inlet} are the outlet and inlet A concentrations, V is reaction volume, A_o the pre-exponential factor, Ea the activation energy, T temperature, and R the universal gas constant.

If F_e is considered constant, the nonlinear term includes the exponential and C_A. Linearizing around the steady state [T0 C_A0].

```
syms  T T0 CA CA0 V R Ea Ao
F=-Ao*exp(-Ea/R/T)*CA*V;
F_approach=taylor (F,[T CA],[T0 CA0],'order',2)
simplify(F_approach)

OUT
F_approach  =-(Ao*V*exp(-Ea/(R*T0))*(CA*R*T0^2  -  CA0*Ea*T0  +
CA0*Ea*T))/(R*T0^2)
```

1.2. Time invariant first order processes

This type of system obeys a differential equation of the formula (Ogata, 2010):

$$a\frac{dy}{dt}+y=b_1u_1+b_2u_2+....+b_mu_m \qquad [1.11]$$

Where a and b_i are constants, y is the output variable while u_i refers to the input variables influencing y.

The simplest case with only one input variable is represented by:

$$\tau\frac{dy}{dt}+y=K_{y,u}u \qquad [1.12]$$

Where τ and $K_{y,u}$ are known as the first order time constant and the gain of the output y regarding the input u. The input u can be any signal, although one of the simplest and most used is the step:

$$u(t)=\begin{cases}\bar{u} & t<0\\ \bar{u}+\Delta u & t\geq 0\end{cases} \qquad [1.13]$$

In MATLAB, as mentioned previously, ordinary differential equations are solved by the command *dsolve*. Symbolic notation must be used to obtain the solution expression.

Example 1.4. Solve a first order differential equation according to equation [1.12] when the initial conditions are set to zero.

```
syms y(t) K u Tau
eqn = Tau*diff(y,t)+y == K*u; % Differential equation
cond = y(0) == 0; % Initial condition
disp('The solution of the first order differential equation is:')
y = dsolve(eqn,cond)

OUT
The solution of the first order differential equation is:
y = K*u - K*u*exp(-t/Tau)
```

The influence of the gain (K) and time constant (τ) are depicted in figure 1.2. when the input signal is a unit step. A value of τ = 2 has been used (variable Tau in the code) when the influence of K is analyzed. A value of K=1 has been used when the influence of τ is considered. The loop generated by the command *for…end* allows the change in the variable to be studied.

```
t = [0:.3:15]; % time vector from 0 to 15 in steps of 0.3
u=1;Tau=2;
figure
hold on
for K=1:10 % gain vector evaluated from 1 to 10 in steps of 1.0
   y=K*u*(1-exp(-t/Tau));
   plot(t,y,'DisplayName', ['K = ' num2str(K)])
   title('Influence of K')
   xlabel('Time')
   ylabel('y(t)')
end
legend('Location', 'best');
hold off
figure
hold on
K=1;
for Tau=1:10 % constant time vector evaluated from 1 to 10 in steps of 1.0
   y=K*u*(1-exp(-t/Tau));
   plot(t,y, 'DisplayName', ['\tau = ' num2str(Tau)])
   title('Influence of Tau')
   xlabel('Time')
```

```
    ylabel('y(t)')
end
legend('Location', 'best');
hold off
```

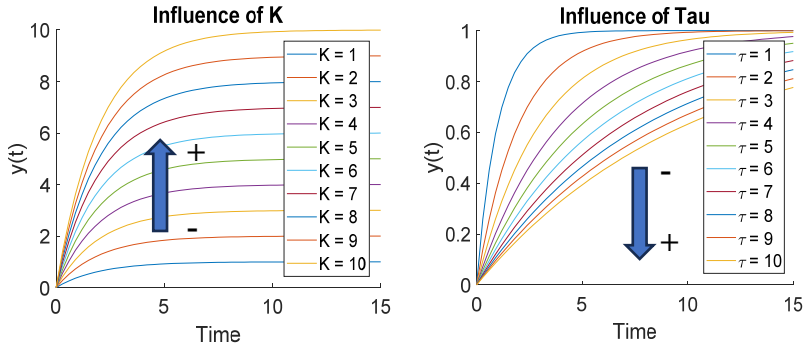


Figure 1.2. Time response of a first order system to a unit step input. Influence of process gain (left) and time constant (right). Variation in parameters from 1.0 to 10.0.

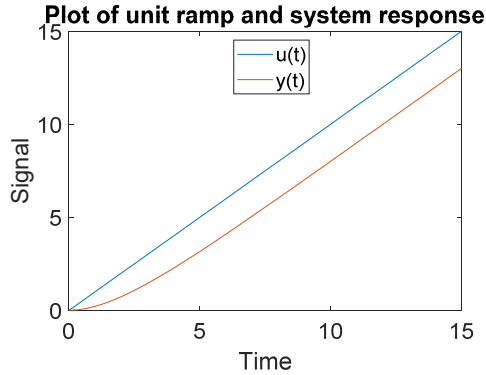
Process gain refers to the factor by which the input signal is multiplied in the system. Higher process gain amplifies the input signal, leading to a more pronounced output response. The time constant represents the time required for the system to reach approximately 63.2% of its final steady-state value when a step input is considered, or alternatively, the time needed to reach the new steady-state if the rate of convergence to it remained consistently equal to the initial rate.

Apart from the step input, other signals can be used, for example the ramp, impulse, rectangular pulse, etc.

Example 1.5. Solve a first order differential equation according to eq. [1.12] when the initial conditions are set to zero and the input signal is a unit ramp u(t) = t.

```
syms y(t) K u Tau
u=t; % Input signal RAMP
eqn = Tau*diff(y,t)+y == K*u; % Differential equation
cond = y(0) == 0; % Initial condition
disp('The solution of the first order differential equation is:')
y = dsolve(eqn,cond)
```

```
time = [0:.3:15];
y=subs(y,[Tau K],[2 1]);% assigned values to Tau=2; K=1;
y=(subs(y,t,time)); % time vector substitution
plot(time,time,time,y)
title('Plot of unit ramp and system response')
xlabel('Time')
ylabel('Signal')
legend('u(t)','y(t)')
```



OUT

y = K*t - K*Tau + K*Tau*exp(-t/Tau)

1.3. Time invariant second order processes

Time invariant second order systems follow differential equations of the type (Ogata, 2010):

$$a_2 \frac{d^2y}{dt^2} + a_1 \frac{dy}{dt} + y = b_1u_1 + b_2u_2 + ... + b_mu_m \qquad [1.14]$$

Again, a_i and b_i are constants, y is the output variable while u_i refers to input variables influencing y. Second order systems are normally expressed by using the following notation:

$$\tau^2 \frac{d^2y}{dt^2} + 2\delta\tau \frac{dy}{dt} + y = K_{y,u}u \qquad [1.15]$$

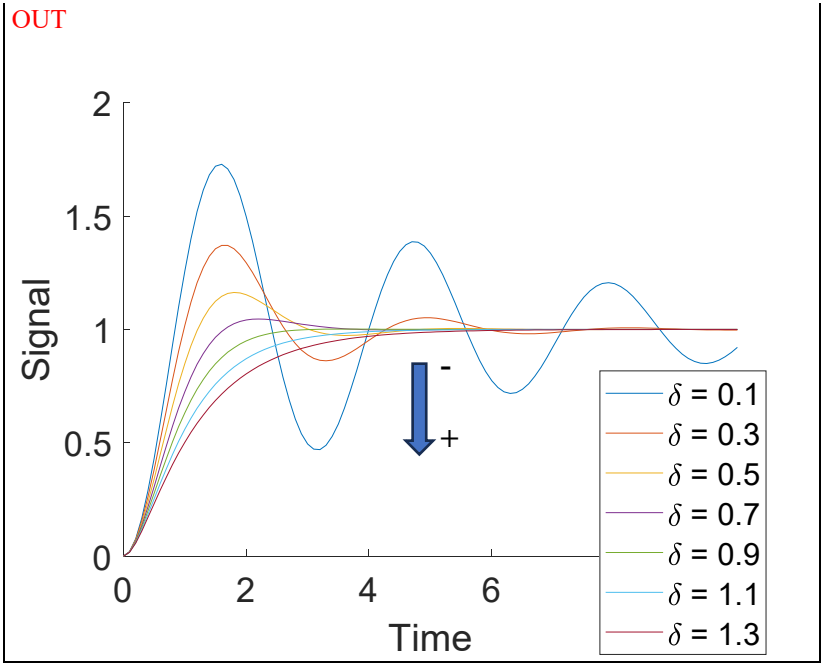
The solution of eq. [1.14] is obtained by using the same commands as in the case of first order systems. In [1.15], τ is the inverse of the natural frequency (ω_n), which is a measure of how quickly the system oscillates in the absence of damping. It represents the rate at which the system completes one full

cycle of oscillation. The damping ratio (δ), on the other hand, determines the decay rate of oscillations in an underdamped second-order system. A lower damping ratio results in slower decay and more pronounced oscillations, while a higher damping ratio leads to faster decay and less pronounced oscillations.

Example 1.6. Solve and represent the response of a time invariant second order system to a unit step input by changing the damping factor from 0.1 to 1.3.

```
syms y(t) K u Tau delta
u = 1; % input signal = unit step
eqn = Tau^2*diff(y,t,2) + 2*Tau*delta*diff(y,t) + y == K*u; %
differential equation
Dy = diff(y,t);
cond = [y(0)==0, Dy(0)==0]; % Initial conditions
disp('Solution of the second order differential equation:')
y = dsolve(eqn,cond);
pretty(y)
time = 0:0.1:10;
figure;
hold on;
for delta_value = 0.1:0.2:1.3
% assigns values for Tau=0.5; K=1; delta=delta_value
% if delta_value =1, the value is slightly modified to avoid division by
zero
if delta_value == 1
      y_value = subs(y, [Tau, K, delta], [0.5, 1, delta_value-0.001]);
   else
      y_value = subs(y, [Tau, K, delta], [0.5, 1, delta_value]);
 end
   y_time = subs(y_value, t, time); % Time substitution
   plot(time, y_time)
end
hold off;
title('2nd order response as a function of damping factor')
xlabel('Time')
ylabel('Signal')
legend('\delta = 0.1',  '\delta = 0.3',  '\delta = 0.5',  '\delta = 0.7', '\delta =
0.9',  '\delta = 1.1',  '\delta = 1.3')
```

OUT



In the previous code, the command *subs (f, old, new)* creates a modified copy of *f* by replacing all occurrences of *old* with *new*, and then evaluates the modified expression f. In this context, *f* represents an expression involving symbolic scalar variables or a symbolic function, and *old* specifies the symbolic scalar variables or symbolic function that are to be substituted with *new*.

From example 1.6, two different type profiles are experienced differentiating between underdamped ($\delta<1$) and overdamped systems ($\delta>1$).

CHAPTER 2

THE LAPLACE DOMAIN

2. Laplace domain system dynamics

The Laplace transform provides a powerful and efficient technique for solving linear differential equations that capture the dynamic behavior of processes. Moreover, it simplifies the development of input-output models, giving rise to the fundamental concept of the transfer function. By analyzing the transfer function, valuable understanding of how a system behaves when subjected to diverse external influences can be obtained (Nise, 2019).

The definition of the Laplace transform of a function f is given by:

$$f(s)=L\left[f(t)\right]=\int_0^{\infty} f(t)e^{-st}dt \qquad [2.1]$$

In MATLAB, the command to move from time to Laplace domains is *laplace(f)* while the opposite direction is achieved by the command *ilaplace(F)*.

Example 2.1. Determine the Laplace transform and inverse Laplace transform of the following f_i functions.

```
syms w A t s
f1=A*sin(w*t);
f2=A*cos(w*t);
f3=A*t;
f4=A*exp(-w*t);
f5=dirac(t); % Delta Dirac
f6 = A*(heaviside(t) - heaviside(t - 1));  % Pulse function
f7= heaviside(t) % Unit step
F1= laplace(f1)
F2= laplace(f2)
F3= laplace(f3)
F4= laplace(f4)
F5= laplace(f5) % Delta Dirac
F6 = laplace(f6)  % Pulse function
F7 = laplace(f7)  % Unit step
f1= ilaplace(F1)
f2= ilaplace(F2)
f3= ilaplace(F3)
f4= ilaplace(F4)
f5= ilaplace(F5) % Delta de Dirac
f6 = ilaplace(F6)  % Pulse function
f7 = ilaplace(F7)  % Unit step

OUT
F1 = (A*w)/(s^2 + w^2)
F2 =(A*s)/(s^2 + w^2)
F3 =A/s^2
F4 =A/(s + w)
F5 =1
F6 =-A*(exp(-s)/s - 1/s)
F7=1/s

f1 =A*sin(t*w)
f2 =A*cos(t*w)
f3 =A*t
f4 =A*exp(-t*w)
f5 =dirac(t)
f6 =A - A*heaviside(t - 1)
f7=1
```

Several properties of the Laplace transform (Oppenheim *et al.*, 1998) can be visualized in MATLAB such as:

-Linearity:

$$L[f_1(t)+f_2(t)] = L[f_1(t)] + L[f_2(t)] = f_1(s)+f_2(s) \qquad [2.2]$$

```
syms A w s t
f1=3*t-t^2; % Any time function f1
f2=5+6*t*sin(t); % Any time function f2
fsum=f1+f2;
L1=laplace(f1);
L2=laplace(f2);
LSUM_1=L1+L2; % Addition of individual Laplace transforms
LSUM_2=laplace(f1+f2); % Laplace transform of the individual functions
addition
inverse_1=ilaplace(LSUM_1)
inverse_2=ilaplace(LSUM_2)
if inverse_1==inverse_2
   fprintf('Laplace transform linearity is confirmed')
else
   fprintf('Laplace transform linearity is NOT confirmed')
end
```

-Laplace transform of a constant multiplied by a function equals the product of the constant multiplied by the function Laplace transform:

$$L\left[Kf(t)\right]=KL\left[f(t)\right]=Kf(s) \qquad [2.3]$$

```
syms  K s t
f1=K*(t*sin(t)-t^2); % Any time function f1 multiplied by K
L1=laplace(f1);
L2=K*laplace(f1/K);
inverse_1=ilaplace(L1)
inverse_2=ilaplace(L2)
if inverse_1==inverse_2
   fprintf('Laplace transform property is confirmed')
else
   fprintf('Laplace transform property is NOT confirmed')
end
```

- Laplace transform of the function n^{th} derivative is:

$$L\left[\frac{d^n f(t)}{dt^n}\right]=s^n f(s)-s^{n-1}f(0)-s^{n-2}f'(0)-s^{n-3}f''(0)-...-f^{n-1}(0) \qquad [2.4]$$

```
syms f(t) s;
n= 2 % Derivative order, in this case second derivative
Df = diff(f(t),t,n);
F=laplace(Df,t,s)

OUT
F= s^2*laplace(f(t), t, s) - s*f(0) - subs(diff(f(t), t), t, 0)
```

- Laplace transform of a function integration is:

$$L\left[\int_0^t f(t)dt\right]=\frac{f(s)}{s} \qquad [2.5]$$

```
syms t s
f = sin(t)*t^2;  % Function definition
F = laplace(int(f, t, 0, t), t, s); % Laplace transform of the integral
F2=laplace(f, t, s); % Laplace transform of the function
F2=F2/s; % Laplace transform of the integral
inverse_1=ilaplace(F)
inverse_2=ilaplace(F2)
if inverse_1==inverse_2
   fprintf('Laplace transform of integral is confirmed')
else
   fprintf('Laplace transform of integral is NOT confirmed')
end
```

- First shifting theorem. Referred to functions of the type:

$$g(t)= \exp(-at)\, f(t) \qquad [2.6]$$

This theorem states that:

$$L\left[\exp(-at)\, f(t)\right] = f(p) = f(s+a) \qquad [2.7]$$

That is, by assuming p = s+a, the Laplace transform of g(t) is obtained by first transforming f(t) and thereafter substituting *s* by *s+a.*

```
syms f g s t a;
f=3*t+4*sin(t); % Any time dependent function
g=exp(-a*t)*f;
G=laplace(g);
% Proof
F=laplace(f);
G2=subs(F,s,s+a);
if G==G2
   fprintf('First shifting theorem right')
else
    fprintf('First shifting theorem NOT right')
end
```

- Second shifting theorem. Referred to functions delayed t_m units:

$$L\left[f(t-t_m)\right]=e^{-st_m}f(s) \qquad [2.8]$$

```
syms f s t;
tm=3; %Delay
f=heaviside(t - tm)*((t - tm)^2+sin(t-tm)); % Sine + t^2 function delayed
three units
F=laplace(f);
F=simplify (F)
% Proof
tm2=0; %Delay
f2=heaviside(t - tm2)*((t - tm2)^2+sin(t-tm2)); % Sine + t^2 function
delayed three units
F2=laplace(f2);
F2=simplify (F2);
F2=exp(-tm*s)*F2
if F==F2
   fprintf('Second shifting theorem right')
else
    fprintf('Second shifting theorem NOT right')
end
```

-Final value theorem: If the time limit exists and the f(t) Laplace transform is F(s), then:

$$\lim_{t\to\infty} f(t) = \lim_{s\to 0} sF(s) \qquad [2.9]$$

```
syms y(t) K u Tau
u=1; % Unit step input
eqn = Tau*diff(y,t)+y == K*u; % First order differential equation in
deviation variables
cond = y(0) == 0; % Initial condition
disp('The solution of the first order differential equation is:')
y = dsolve(eqn,cond)
F=laplace(y) % Laplace transform
limit(s*F,s,0) % Final value theorem confirmation
```

In the previous code *limit(s*F,s,0)* returns the bidirectional limit of the symbolic expression *s*F* when *s* approaches 0.

-Scaling property in the Laplace domain.

This property states that if F(s) is the Laplace transform of f(t), then the Laplace transform of t^n * f(t) can be obtained by multiplying the original Laplace transform by (-1)^n times the n^{th} derivative of F(s) with respect to s.

$$L\left[t^n\ f(t)\right] = (-1)^n \frac{d^n F(s)}{ds^n} \qquad [2.10]$$

```
syms t s
n = 2;  % Power of time
f_wt =  exp(-2*t)+sin(t);  % Function without power time multiplication
f=  t^n * f_wt; % Function with power time multiplication
F = laplace(f, t, s)  % Laplace transform of the complete function
F_wt = laplace(f_wt, t, s);  % Laplace transform of the function with no
time power
scaledF = (-1)^n * diff(F_wt, s, n)  % Scaling property in the Laplace
domain
```

2.1. -Inverse Laplace transform by Heaviside expansion

The inverse Laplace transform by Heaviside expansion is a method used to find the time domain equations of a function after expressing the Laplace transform in terms of known elementary functions and applying the inverse to each term. The Heaviside expansion method is based on the partial fraction decomposition technique, where the Laplace transform is decomposed into a sum of simpler fractions (Croft and Croft, 2017).

Two commands can be used in MATLAB. Command *[r,p,k] = residue(numerator, denominator)* returns the residues, poles, and constant term of a fractional expression defined by polynomials *numerator* and *denominator*. Also, the command *partfrac(expr,var)* finds the partial fraction decomposition of *expr* with respect to *var*.

Example 2.2. Find the simple fraction expansion of the expression:

$$\frac{(s+8)(s+4)}{\left(s^5 + 10s^4 + 34s^3 + 52s^2 + 37s + 10\right)} \qquad [2.11]$$

Find the inverse Laplace transform from the original equation and from the expansion in simple fractions.

```
% Option 1
syms s
F = (s+8)*(s+4)/(s^5 + 10*s^4 + 34*s^3 + 52*s^2 + 37*s + 10);
% Rational function
[num, den] = numden(F);  % Separate numerator and denominator
num = sym2poly(num);  % Conversion to coefficients of numerator
polynomial
den = sym2poly(den);   % Conversion to coefficients of denominator
polynomial
[r, p, k] = residue(num, den);  % Calculate residues, poles, and constant
term
n=1;
for i = 1:length(r)-1
   if p(i)==p(i+1) || (i~=1 && p(i)==p(i-1))
      disp (['term ', num2str(i),'       ',num2str(r(i)),' /(s- (', num2str(p(i)),'))
^ ', num2str(n)])
       n=n+1;
      if p(i)~= p(i+1)
         n=1;
      end
   else
      n=1;
      disp (['term ', num2str(i),'       ',num2str(r(i)),' /(s- (', num2str(p(i)),'))
^ ', num2str(n)])
   end
end
if p(length(r))==p(length(r)-1)
```

```
   disp (['term ', num2str(length(r)),'       ',num2str(r(length(r))),' /(s- (',
num2str(p(length(r))),')) ^ ', num2str(n)])
else
    disp (['term ', num2str(length(r)),'       ',num2str(r(length(r))),' /(s- (',
num2str(p(length(r))),')) ^ ', num2str(1)])
end
% Option 2
SumTerm= partfrac(F,s,'FactorMode','full')
% Inverse Laplace transforms, checking the analogy
inverse_1=ilaplace(F)
inverse_2=ilaplace(SumTerm)

OUT
term 1       -0.015625 /(s- (-5)) ^ 1
term 2       -4 /(s- (-2)) ^ 1
term 3       4.0156 /(s- (-1)) ^ 1
term 4       -4.0625 /(s- (-1)) ^ 2
term 5       5.25 /(s- (-1)) ^ 3
SumTerm =257/(64*(s + 1)) - 65/(16*(s + 1)^2) - 4/(s + 2) + 21/(4*(s +
1)^3) - 1/(64*(s + 5))
inverse_1 =(257*exp(-t))/64 - 4*exp(-2*t) - exp(-5*t)/64 - (65*t*exp(-
t))/16 + (21*t^2*exp(-t))/8
inverse_2 =(257*exp(-t))/64 - 4*exp(-2*t) - exp(-5*t)/64 - (65*t*exp(-
t))/16 + (21*t^2*exp(-t))/8
```

2.2. Differential equations systems solution by Laplace transform

The solution of differential equations systems can be achieved by first applying the Laplace transform and thereafter solving the resulting algebraic system.

Example 2.3. Find the solution of the differential equations system represented in deviation variables:

$$\begin{aligned} \frac{dx_1}{dt} &= 2x_1 + 3x_2 + 1 \\ \frac{dx_2}{dt} &= 2x_1 + x_2 + e^t \end{aligned} \qquad [2.12]$$

Compare the results with those obtained by using the *dsolve* command.