In the name of God

Univrsity of Tehran

Faculty of engineering

Electrical and computer faculty

**Linear control systems lab**

**Exp2**

—

--

Group numb:5

Aban 1398

**Contents**

|  |  |
| --- | --- |
| **Title** | **Page number** |
| Abstract | 3 |
| Part 2 | 4 |
| Ps | 9 |
| References | 10 |

Abstract

Dc motor

In this experiment we want to simulate a DC motor in matlab so we can control its speed and angle of deviation. if we want to build a real DC motor we have to first simulate their values in matlab and after being insured about outputs we can fit our values to DC motor.

Two kinds of controlling DC motors:

Controlling armature current

Magnetic field fluid

We have to notice that in the simulation part we assume that out DC motor is ideal.

For evaluating control ability of our system we can give it step function as an input and by changing its poles and zeros find the stable point in our system. In matlab simulation we can build our blocks with their conservation function and then connect them as parallel or series and after being completed we can make our DC motor by making one block with specific inputs and outputs.

# Part 2

## 2-1. simulation of differential equations in matlab/Simulink



Graph I

Graph I shows Step responses of ,, (from Simulink)

## 2-2. Simulation of differential equations in matlab editor

### 2-2-1. Simulation with conservation function



Graph II

Graph II shows Step responses of ,, (from Matlab codes)

Comparing Graph II with Graph I:

Speed is proportional to which is proportional to .

Position is proportional to .

Input current is our armature current.

### 2-2-2. Simulation with differential equation



Graph III

Graph III shows the solution for differential equation of 𝑥̇ with solver Ode45.

𝑥̇ = −𝑥2𝑒−0.1𝑡 sin(𝑡) + 10𝑒−5𝑡; 𝑥(0) = 0



Graph IV

Graph IV shows the solution for differential equation (1) with 1 (voltage) as an input and {position, speed , armature current } as our output. With ‘0’ initial condition and for 5s .

## Home research (Reference, n.d.)

## 

Ode1:

Use the Ode1 solver with a first order of accuracy.

Ode3:

Use Ode3 solver with a third order of accuracy.

Ode45:

matlab’s standard solver for ordinary differential equations (ODEs) is the function Ode45. This function implements a Runge-Kutta method with a variable time step for efficient computation.

Ode23:

Compares method of order two and three to automatically choose the step size and maintain a specified accuracy. It is the simplest matlab solver that has modern features such as automatic error estimate and continues interpolant. Ode23 is suitable for coarse accuracy requirements such as computer graphics.

Ode113:

Problem type: non shift, order of accuracy: low to high, when to use: If using stringent error tolerances or solving a computationally intensive ODE file.

Ode15s:

Problem type: stiff, order of accuracy: low to medium, when to use: if Ode45 is slow because the problem is stiff.

Ode23s:

Problem type: stiff, order of accuracy: low, when to use: if using crude error tolerances to solve stiff systems and the mass matrix is constant.

Ode23t:

Problem type: moderately stiff, order of accuracy: low, when to use: if the problem is only moderately stiff and you need a solution without numerical damping.

Ode23tb:

Problem type: stiff, order of accuracy: low, when to use: if using crude error tolerance to solve stiff systems.

OdeN:

The ordinary differential equation (ODE) solvers in matlab solve initial value problems with a variety of properties. The solvers can work on stiff or non-stiff problems, problems with a mass matrix, differential algebraic equations (DAEs), or fully implicit problems.

# Ps1: process of program

We have three codes file called p2\_2\_1 & p2\_2 & p2\_4. in the first code file p2\_2\_1 we write conservation function of speed with respect of voltage and as well as conservation function for position. In addition of those function we have conservation function of armature current with respect of voltage and at the end we give them step function as input.

In p2\_2 we return our Simulink values into matlab workspace and plot them.

In p2\_4 we solve our equation, differential equation with matlab solver methods such as Ode45.

# References

D.

MATLAB Function Reference, Oct 31, 2019.

Accessed on: Oct 31, 2019. [Online].

Available: http://www.ece.northwestern.edu/local-apps/matlabhelp/techdoc/ref/ode45.html