Lecture 0: Preliminaries

Erfan Nozari

September 24, 2022

Welcome to ME 120! This course provides you with the basics of linear systems analysis and control and is aimed to catalyze, at an introductory level, between pure math and engineering applications.

In this note I brush up some of the basic mathematical concepts that we need to begin with: differential equations, complex numbers, and some MATLAB. Remember that I am assuming you have learned these in related courses before, and here only aim to briefly review them, so feel free to make a pass at your old notes and textbooks if you feel the need to!

Contents

0.1 Ordinary Differential Equations (ODEs) . . . 1
- 0.1.1 Notation (Time derivative) . . . 2
- 0.1.2 MATLAB (Numeric vs. symbolic math) . . . 3
- 0.1.3 MATLAB (Solving first-order ODEs) . . . 3
- 0.1.4 Exercise (Solving more complex ODEs) . . . 4

0.2 Complex Numbers . . . 4
- 0.2.1 Notation (Real and complex numbers) . . . 5
- 0.2.2 Definition (Complex exponential) . . . 5
- 0.2.3 MATLAB (“for” loop and exponential function) . . . 5
- 0.2.4 Definition (Cartesian and polar forms) . . . 6
- 0.2.5 Definition (Complex number attributes) . . . 6
- 0.2.6 MATLAB (Complex attributes) . . . 6
- 0.2.7 Theorem (Fundamental theorem of algebra) . . . 7
- 0.2.8 Exercise (Roots of cubic polynomial) . . . 7
- 0.2.9 MATLAB (Roots of polynomials) . . . 7

0.1 Ordinary Differential Equations (ODEs)

Remember that an ODE is a mathematical algebraic relationship between a function, say $x(t)$, and its derivatives. The independent variable t does not need to be time, but in this course we are only interested

Copyright © 2022 by Erfan Nozari. Permission is granted to copy, distribute and modify this file, provided that the original source is acknowledged.

in derivatives with respect to time. Some toy examples are

$$\dot{x}(t) = -x(t) + 2\cos(t) \tag{0.1a}$$

$$\cos\left(x^3(t) - e^{\dot{x}(t)}\right) - \sin(8t) = 0 \tag{0.1b}$$

$$\ddot{x}(t) - \dot{x}(t) + 2x(t) = te^{-t} \tag{0.1c}$$

but there are also extensively used real-world examples all over science and engineering, such as

Newton's 2nd law: $$u(t) = m\ddot{x}(t) \tag{0.2a}$$

Harmonic oscillator: $$\ddot{x}(t) = -\alpha x(t) \tag{0.2b}$$

Van der Pol's oscillator: $$\ddot{x}(t) + \mu(x^2(t) - 1)\dot{x}(t) + x(t) = 0 \tag{0.2c}$$

Hodgkin-Huxley neuron model: $$u(t) = C\dot{x}(t) + g_{\rm K}(x(t) - V_{\rm K}) + g_{\rm Na}(x(t) - V_{\rm Na}) + g_{\rm L}(x(t) - V_{\rm L}) \tag{0.2d}$$

Chemical reactions: $$\dot{x}(t) = \gamma_1 k_1 x(t)^a u(t)^b + \gamma_2 k_2 x(t)^c u(t)^d \tag{0.2e}$$

Notation 0.1.1 (Time derivative) As you have noticed, we use dots to show derivatives, so

$$\dot{x}(t) = \frac{d}{dt}x(t)$$

$$\ddot{x}(t) = \frac{d^2}{dt^2}x(t)$$

and so on. □

In all the ODEs in Eq. (0.2), the constant variables $m, \alpha, \mu, C, ...$ are parameters. More important is the function $u(t)$ of time, which represents "external inputs" to the ODE. In general $u(t)$ may be any function of time, such as

$$u(t) = 0$$

$$u(t) = \alpha e^{\beta t}$$

$$u(t) = A\cos(\omega t + \theta_0)$$

$$\vdots$$

Solving an ODE means finding a function (or functions) $x(t)$ that satisfy the ODE. The function $x(t) = e^{-t} + \sqrt{2}\cos(t - \pi/4)$, for example, satisfies Eq. (0.1a) (check it!). But so does $x(t) = ke^{-t} + \sqrt{2}\cos(t - \pi/4)$ for any number k. This is why solving an ODE for a *unique solution* needs more information, typically provided as "initial conditions"

$$x(t_0) = x_0$$

$$\dot{x}(t_0) = \dot{x}_0$$

$$\vdots$$

where t_0 is some "initial time" (often 0) and $x_0, \dot{x}_0, \ldots$ are given numbers. The number of initial conditions needed to solve an ODE for a unique solution is typically as many as the largest order of derivative in the ODE.

As you might remember from your course in differential equations, not all ODEs can be analytically solved, and solving them is quite some art! Fortunately, we here only care about linear ODEs, which have various systematic method to solve. But before getting there, let's see how we can solve (linear and nonlinear) ODEs using MATLAB.

Copyright © 2022 by Erfan Nozari. Permission is granted to copy, distribute and modify this file, provided that the original source is acknowledged.

MATLAB 0.1.2 (Numeric vs. symbolic math) When using MATLAB, you have two main options:

- Numeric calculations
- Symbolic calculations

As the name suggests, numeric calculations use real (or complex) numbers, while symbolic calculations try to mimic hand-written formulas where we mix numbers and symbols. For example, try

```
1 syms x y
2 expand((x + y)^2)
```

The `syms` command defines symbolic variables, and the `expand` command *tries* to expand a symbolic expression into simpler terms. Another useful command when using symbolic math is `simplify`. For example, try

```
1 syms theta
2 simplify(sin(theta)^2 + cos(theta)^2)
```

or

```
1 syms a b
2 simplify((a+b)^2 - 2*a*b)
```

As you see, it *tries* to simplify the expression by cancelling out terms, applying trigonometric identities, etc. Finally, when using symbolic math, it's nice to visualize expressions the way we write them on paper. For example, try

```
1 % Assuming that you have defined x and y above
2 f = sqrt(x^2 + 1) / ((x^3 + 1)^2 - 2*x*y)
3 pretty(f)
```

□

MATLAB 0.1.3 (Solving first-order ODEs) ODEs can be solved both numerically and symbolically in MATLAB, and both have their advantages and disadvantages. Symbolic solutions are perfectly exact, but they only exist for very simple ODEs, whereas numeric solutions are approximates but computing them is usually faster and they exist for virtually all ODEs.

You can use `dsolve()` to solve ODEs analytically and `ode45()` to solve them numerically. For example, to solve Eq. (0.1a) with $x(0) = 2$ analytically, use

```
1 syms t x(t)
2 sol = dsolve(diff(x) == -x + 2*cos(t), x(0) == 2)
```

In the first line, we are defining `x` as a symbolic *function* of `t`. Note that you don't have to define the symbolic variable `t` separately. The second line solves the ODE. Try removing the initial condition and see how the solution changes:

```
1 % Assuming that you have defined x(t) above
2 sol = dsolve(diff(x) == -x + 2*cos(t))
```


Copyright © 2022 by Erfan Nozari. Permission is granted to copy, distribute and modify this file, provided that the original source is acknowledged.

You see that the solution includes a constant, because without initial conditions, the ODE has infinitely many solutions.

Now we solve the same equation numerically. Here, we have to provide the initial condition because everything has to be in numbers (no symbolic variables anymore). Even more, we have to specify the exact time interval over which we want the solution. So, try

```
odefun = @(t, x)-x + 2*cos(t);
tinterval = [0 100];
x0 = 2;
[tspan, x] = ode45(odefun, tinterval, x0);
```

The first line is the most confusing part. This is MATLAB syntax for defining what is called a "function handle". So when you run the first line, MATLAB defines the function $f(t, x) = -x + 2\cos(t)$ for you internally, and returns a "handle" to it called `odefun` which you can use to call that function. So after running the first line, you can run

```
odefun(0, 2)
odefun(pi/2, 2)
odefun(pi, 3)
```

and so on. So I want you to realize that the first line by itself has nothing to do with solving ODEs. It is just a way of defining functions in MATLAB, and one application of it is in solving ODEs. The second line above defines the interval of time over which you want to solve the ODE, and the third line defines the initial condition. The ODE is solved in the last line, when you call the MATLAB function `ode45`. The second output is the solution, and the first output is a list of time points corresponding to `x`. To visualize the solution, try

```
figure
plot(tspan, x)
```

The first line opens an empty figure, and the second line plots `tspan` on the horizontal axis and `x` on the vertical axis. □

Exercise 0.1.4 (Solving more complex ODEs) Try solving Eq. (0.1b) both symbolically and numerically in MATLAB. Visualize your results for different initial conditions. □

0.2 Complex Numbers

Complex numbers are essential in the linear systems theory because of their role in finding the roots of polynomial functions (to be discussed shortly). Recall (or learn!) that a complex number is nothing but an ordered pair (x, y) of real numbers $x, y \in \mathbb{R}$, with a specific rule for multiplication such that

$$(x_1, y_1) \cdot (x_2, y_2) = (x_1x_2 - y_1y_2, x_1y_2 + x_2y_1) \tag{0.3}$$

For simplicity, we then define

$$j = (0, 1)$$

as the imaginary unit number, and take complex number $(x, 0)$ equivalent to the real number x. Then, it is easy to show that any complex number (x, y) can also be represented as $x + jy$ (check yourself!), which is

Copyright © 2022 by Erfan Nozari. Permission is granted to copy, distribute and modify this file, provided that the original source is acknowledged.

what we often know as complex numbers. It also follows from Eq. (0.3) that

$$j^2 = j \cdot j = (0,1) \cdot (0,1) = (-1,0) = -1$$

or, as commonly known, $j = \sqrt{-1}$.

One of the most important elements in complex analysis is the complex exponential. Recall that the real exponential e^x is defined though the infinite sum

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

and has the property that $e^{x+y} = e^x e^y$.

Notation 0.2.1 (Real and complex numbers) You have noticed that we use $\mathbb{R}$ to show the set of real numbers. We will similarly use $\mathbb{C}$ for the set of complex numbers. □

Definition 0.2.2 (Complex exponential) For a complex number $z = x+jy \in \mathbb{C}$, the complex exponential is defined using the same sum as the real exponential,

$$e^z = \sum_{k=0}^{\infty} \frac{z^k}{k!} \tag{0.4}$$

and has the property that, for $x \in \mathbb{R}$,

$$e^{jx} = \cos x + j \sin x$$

which is called the Euler's formula. □

MATLAB 0.2.3 ("for" loop and exponential function) One of the most important concepts in programming are loops, and we can learn/recall them here to better understand Eq. (0.4). Recall that the infinite series in Eq. (0.4) means that if K is large enough, then

$$e^z \simeq \sum_{k=0}^{K} \frac{z^k}{k!} = 1 + z + \frac{z^2}{2} + \cdots + \frac{z^K}{K!}$$

So try the following for different complex values for `z` and integer values for `K`:

```
1 z = 1 + 2j;
2 K = 20;
3 finite_sum = 0;
4 for k = 0:K
5     finite_sum = finite_sum + z^k / factorial(k);
6 end
7 error = abs(exp(z) - finite_sum)
```

□

Similar to the real exponential, the complex exponential satisfies $e^{z_1+z_2} = e^{z_1}e^{z_2}$. Therefore,

$$e^{x+jy} = e^x e^{jy} = e^x \cos y + je^x \sin y$$

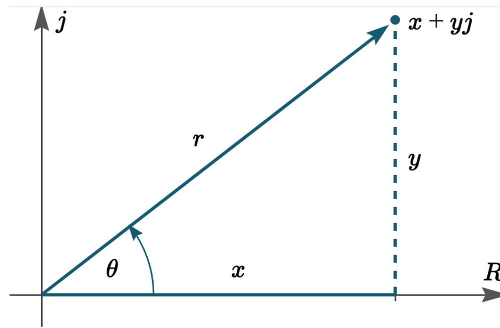
This motivates a very useful transformation in complex numbers, that to "polar coordinates".

Copyright © 2022 by Erfan Nozari. Permission is granted to copy, distribute and modify this file, provided that the original source is acknowledged.

Definition 0.2.4 (Cartesian and polar forms) The complex number $z = x+jy$ is said to be in Cartesian coordinates, since it can be drawn as the point (x, y) on the Cartesian plane (shown below). Let (r, θ) be the representation of the same point in the plane in polar coordinates, as shown. Then

$$z = x + jy = r\cos\theta + jr\sin\theta = r\big(\cos\theta + j\sin\theta\big) = re^{j\theta}$$

The representation $z = re^{j\theta}$ is called the polar form of z.



□

We are now ready for a few more fundamental definitions about complex numbers.

Definition 0.2.5 (Complex number attributes) For a complex number $z = x + jy = re^{j\theta}$,

- x is called the real part of z, shown as $\mathrm{Re}\{z\}$;
- y is called the imaginary part of z, shown as $\mathrm{Im}\{z\}$;
- r is called the modulus of z, shown as $|z|$, and equal to $\sqrt{x^2 + y^2}$; note that

$$|e^{j\theta}| = \sqrt{\cos^2\theta + \sin^2\theta} = 1$$

- θ is called the argument of z, shown as $\measuredangle z$
- $\bar{z} = x - jy = re^{-j\theta}$ is called the complex conjugate of z.

□

MATLAB 0.2.6 (Complex attributes) You can transform complex numbers from Cartesian to polar coordinates and vice versa, and compute real part, etc. For example,

```
1  z = 3 - 4j;
2  x = real(z)
3  y = imag(z)
4  r = abs(z)
5  theta = angle(z) % in radians
6  theta = angle(z) * 180 / pi % in degrees
7  [theta, r] = cart2pol(x, y) % should get same theta and r as above
8  [x, y] = pol2cart(theta, r) % should get same x and y as above
```

□

As I mentioned earlier, our main reason to dealing with complex numbers in linear systems and control is because of their role in the roots of polynomial functions. The following theorem shows how.

Copyright © 2022 by Erfan Nozari. Permission is granted to copy, distribute and modify this file, provided that the original source is acknowledged.

Theorem 0.2.7 (Fundamental theorem of algebra) Consider the polynomial function

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0.$$

where the coefficients $a_0, \ldots, a_n$ are complex numbers and not all equal to 0. Then, the equation

$$p(z) = 0 \tag{0.5}$$

has exactly n (potentially repeated) complex solutions called the roots of the polynomial p. □

Note the difference between real roots and complex roots. For example, $p(x) = x^2 + k, x \in \mathbb{R}$ has two solutions only if $k < 0$, one solution if $k = 0$, and no solutions if $k > 0$, but $p(z) = z^2 + k, z \in \mathbb{C}$ always has exactly two solutions.

In this course, we only care about the cases where the coefficients $a_0, \ldots, a_n$ are real. In this case, it is not hard to show that all the solutions to Eq. (0.5) are either real, or if they are complex, they come in complex conjugate form. In other words, if $p(z) = 0$, then necessarily $p(\bar{z}) = 0$ as well.

Exercise 0.2.8 (Roots of cubic polynomial) How many of the roots of $p(z) = a_3 z^3 + a_2 z^2 + a_1 z + a_0$ can be complex? □

MATLAB 0.2.9 (Roots of polynomials) Numerically, use the function `roots()` to find the roots of a polynomial. Provide only the polynomial coefficients to the function. For example, for

$$\begin{aligned} p(z) &= z^5 - 2z^4 + 3z^2 \\ &= z^5 - 2z^4 + 0z^3 + 3z^2 + 0z + 0 \end{aligned}$$

you use

```
1 p = [1 -2 0 3 0 0];
2 z = roots(p)
```

Alternatively, define $p(z)$ as a symbolic expression and use `solve` to obtain its roots analytically (if possible)

```
1 syms z
2 p(z) = z^5 - 2 * z^4 + 3 * z^2;
3 sol = solve(p == 0)
```

To see what happens if the solutions don't have a closed-form expression, try solving $p(z) = 3z^5 - 5z^4 + 3z^2$.
□

Copyright © 2022 by Erfan Nozari. Permission is granted to copy, distribute and modify this file, provided that the original source is acknowledged.