Numerical Simulation of Dynamic Systems I

Prof. Dr. François E. Cellier
Department of Computer Science
ETH Zurich

February 26, 2013

A Circuit Example

Given the electrical circuit:

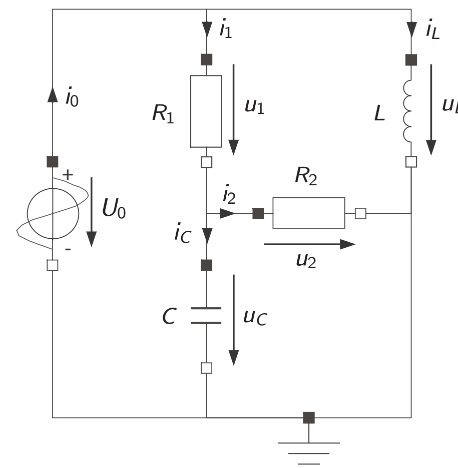


Figure: Circuit diagram of electrical RLC circuit

Implicit Differential Algebraic Equation (DAE) Model

Constitutive equations:

$$\begin{aligned} u_0 &= 10 \\ u_1 - R_1 \cdot i_1 &= 0 \\ u_2 - R_2 \cdot i_2 &= 0 \\ i_C - C \cdot \frac{du_C}{dt} &= 0 \\ u_L - L \cdot \frac{di_L}{dt} &= 0 \end{aligned}$$

Mesh equations (Kirchhoff's Voltage law - KVL):

$$\begin{aligned} u_0 - u_1 - u_C &= 0 \\ u_L - u_1 - u_2 &= 0 \\ u_C - u_2 &= 0 \end{aligned}$$

Node equations (Kirchhoff's current law - KCL):

$$\begin{aligned} i_0 - i_1 - i_L &= 0 \\ i_1 - i_2 - i_C &= 0 \end{aligned}$$

Explicit DAE Model

We can causalize the equations (for now, we won't discuss, how this is being done):

$$\begin{aligned} u_0 &= 10 \\ u_2 &= u_C \\ i_2 &= \frac{1}{R_2} \cdot u_2 \\ u_1 &= u_0 - u_C \\ i_1 &= \frac{1}{R_1} \cdot u_1 \\ u_L &= u_1 + u_2 \\ i_C &= i_1 - i_2 \\ \frac{di_L}{dt} &= \frac{1}{L} \cdot u_L \\ \frac{du_C}{dt} &= \frac{1}{C} \cdot i_C \\ i_0 &= i_1 + i_L \end{aligned}$$

Ordinary Differential Equation (ODE) Model

By substitution, we can eliminate the algebraic equations:

State equations:

$$\frac{du_C}{dt} = -\frac{R_1 + R_2}{R_1 \cdot R_2 \cdot C} \cdot u_C + \frac{1}{R_1 \cdot C} \cdot u_0$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_0$$

Output equation:

$$i_2 = \frac{1}{R_2} \cdot u_C$$

Such a model is often referred to as a *state-space model*.

Linear State-space Model

If the state-space model is linear, as in the given case, it can be written in a matrix-vector form:

$$\begin{pmatrix} \frac{du_C}{dt} \\ \frac{di_L}{dt} \end{pmatrix} = \begin{pmatrix} -\frac{R_1+R_2}{R_1 \cdot R_2 \cdot C} & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} u_C \\ i_L \end{pmatrix} + \begin{pmatrix} \frac{1}{R_1 \cdot C} \\ \frac{1}{L} \end{pmatrix} \cdot u_0$$
$$i_2 = \begin{pmatrix} \frac{1}{R_2} & 0 \end{pmatrix} \cdot \begin{pmatrix} u_C \\ i_L \end{pmatrix}$$

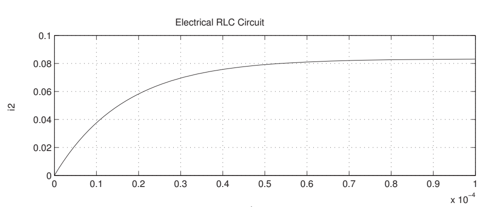
The variables u_C and i_L are called *state variables*, and the vector consisting of the state variables constitutes the *state vector*. In general:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{u} \quad ; \quad \mathbf{x} \in \mathcal{R}^n \; ; \; \mathbf{u} \in \mathcal{R}^m \; ; \; \mathbf{y} \in \mathcal{R}^p$$
$$\mathbf{y} = \mathbf{C} \cdot \mathbf{x} + \mathbf{D} \cdot \mathbf{u} \quad ; \quad \mathbf{A} \in \mathcal{R}^{n \times n} \; ; \; \mathbf{B} \in \mathcal{R}^{n \times m} \; ; \; \mathbf{C} \in \mathcal{R}^{p \times n} \; ; \; \mathbf{D} \in \mathcal{R}^{p \times m}$$

MATLAB Simulation Program

```
% Enter parameter values
%
R1 = 100;
R2 = 20;
L = 0.0015;
C = 1e-6;
%
% Generate system matrices
%
R1C = 1/(R1 * C);
R2C = 1/(R2 * C);
a11 = -(R1C + R2C);
A = [ a11 , 0 ; 0 , 0 ];
b = [ R1C ; 1/L ];
c = [ 1/R2 , 0 ];
d = 0;
%
% Make a system and simulate
%
S = ss(A, b, c, d);
t = [ 0 : 1e-6 : 1e-4 ];
u = 10 * ones(size(t));
x0 = zeros(2, 1);
y = lsim(S, u, t, x0);
%
```

```
% Plot the results
%
subplot(2, 1, 1)
plot(t, y, 'k-')
grid on
title(Electrical RLC Circuit)
xlabel(time)
ylabel(i2)
print -deps fig1_2.eps
return
```



Modeling vs. Simulation

- The transition from the graphical model representation (in our example, a circuit diagram) to the executable MATLAB code is long and cumbersome, even for such a trivial example as the one presented.
- This process needs to be automated.
- For this, we have tools such as *Dymola*. In fact, the circuit diagram shown at the beginning *is* a Dymola program.
- However, it is not the aim of this class to discuss that transition. For this, we provide a second class, entitled **Mathematical Modeling of Physical Systems**, which is offered in the fall semester.
- The purpose of the class **Numerical Simulation of Dynamic Systems** is to discuss the properties of numerical ODE solvers and their codes, as well as the algorithms behind these solvers.

Time and Again

When we simulate a continuous-time system on a digital computer, some quantity will have to be discretized, as we cannot update the state variables infinitely often within a finite time period. Most numerical ODE solvers discretize the time axis, i.e., they advance the *simulation clock* using finite *time steps*. The time step, h, may either be fixed or variable.

We notice in the MATLAB code shown earlier the statement:

```
t = [ 0 : 1e-6 : 1e-4 ];
```

However, 10^{-6} is not the time step, but rather the *communication interval*. With this statement, we instruct the program to report the simulation results back once every 10^{-6} time units.

If a time step passes through a *communication point*, some numerical ODE solvers will reduce their time step to hit the communication point precisely, whereas others will simulate across with the full step size and then interpolate back to report the state vector at the desired time instant.

Time and Again II

- The step size, h, is not necessarily identical with the time advance, Δt, of model evaluations. Many integration algorithms, such as the famous *Runge-Kutta algorithms*, perform multiple model evaluations within a single time step. Thus, each time step, h, contains several micro-steps, Δt, whereby Δt is not necessarily a fixed divider of h. Instead, the simulation clock may jump back and forth within each individual time step.
- Even if the integration algorithm used is such that Δt remains positive at all times, the simulation clock does not necessarily advance monotonously with real time. There are two types of error-controlled integration algorithms that differ in the way they handle steps that exhibit an error estimate that is too large. *Optimistic algorithms* simply continue, in spite of the exceeded error tolerance, while reducing the step size for the subsequent step. In contrast, *conservative algorithms* reject the step, and repeat it with a smaller step size. Thus, whenever a step is rejected, the simulation clock in a conservative algorithm turns back to repeat the step, while not committing the same error.

Time and Again III

- Even if an optimistic algorithm with positive Δt values is being employed, the simulation clock may still not advance monotonously with real time. The reason is that integration algorithms cannot integrate across *discontinuities in the model*. Thus, if a discontinuity is encountered somewhere inside an integration step, the step size must be reduced and the step must be repeated, in order to place the discontinuity in between subsequent steps.

One of the important tasks that we shall be dealing with in this class, beside from looking at the different numerical integration algorithms themselves, is to discuss the various time advance mechanisms.