Laboratory 5 Laplace Transforms & Circuits 2; random numbers.

This laboratory covers an introduction to the Laplace transform and continues with circuit analysis from the last laboratory. There is a little more on symbolic maths and a brief section on random numbers.

References regarding circuit analysis are to "Introduction to Electric Circuits", 6th edition, Wiley, 2004, by Dorf and Svoboda (DS).

Solving Differential Equations

The dsolve function computes solutions to ordinary differential equations. An example:

```
>> dsolve('Dy=y')
ans =
C1*exp(t)
```

Note the use of `Dx` for the derivative of `x`. Similarly, `D2x` represents the second derivative, etc. Note also the use of `t` as the default independent variable and of `C1` as an arbitrary constant. We can specify an initial condition and/or change the independent variable:

```
>> dsolve('Dy=y', 'y(0)=5', 'x')
ans =
5*exp(x)
```

If there are several equations in several variables, the solution is returned as a structure:

```
>> S = dsolve('Dx+x=2*y', 'Dy+y=0', 'x(0)=1', 'y(1)=2')
S =
    x: [1x1 sym]
    y: [1x1 sym]
```

The solutions are the fields of the structure `S`. Thus

```
>> S.x
ans =
exp(-t)*(1+4*t/exp(-1))
```

The Laplace Transform

The Symbolic Maths Toolbox has functions to handle the Laplace, Fourier and Z-transforms. In this laboratory, we will consider only the Laplace transform, which is important in linear systems analysis (including circuit analysis). The Laplace transform of the function $f(t)$ is defined by (DS 14.3)

$$F(s) = \int_{0-}^{\infty} f(t)e^{-st}dt$$

Usually, *t* represents time and *s* the complex frequency $s = \sigma + j\omega$. The derivation of the Laplace transform is carried out in MATLAB with the laplace function. Some examples:

```
>> laplace(sym(1))
ans =
1/s
>> laplace(t^3)
ans =
6/s^4
>> laplace(cos(2*t))
ans =
s/(s^2+4)
```

MATLAB uses the variable s by default. The inverse Laplace integral (DS 14.5-the equation is omitted here) is obtained using the function ilaplace. For example:

```
>> syms s
>> ilaplace(1/s)
ans =
1
```

Note that, as can be seen from the defining integral, the value of $f(t)$ for $t < 0$ has no effect on $F(s)$; however, mathematically the formal inverse Laplace transform produces a function $f(t)$ which is zero in this range. The function ilaplace returns a function of *t* which is correct for positive *t* but not expressly zero for negative *t*.

Here are some examples which illustrate some interesting points. Suppose we wish to find the $f(t)$ which has the Laplace transform

$$F(s) = \frac{s}{s^3 + 5s^2 + 16s + 30}.$$

We can use ilaplace:

```
>> syms s
>> F = s/(s^3+5*s^2+16*s+30);
>> f = ilaplace(F);
>> pretty(f)
       /            sin(3 t) 7 \
exp(-t) | cos(3 t) + ---------- | 3
        \                 9     /     exp(-3 t) 3
---------------------------------- - -----------
                13                        13
```

If we now take the Laplace transform of the result:

```
>> F = laplace(f);
>> pretty(F)
                 1             s + 1              1
       - 3/13 ----- + 3/13 ------------ + 7/13 ------------
              s + 3                2                   2
                          (s + 1)  + 9        (s + 1)  + 9
```

we obtain the original function of $F(s)$ as (a form of) partial fractions, each term corresponding to a term of $f(t)$. (Note: you may get a slightly different display if you try this yourself.)

Suppose, however, that

$$F(s) = \frac{8s^2}{16s^3 + 30s^2 + 18s + 3}.$$

When we use ilaplace to find $f(t)$, the result is:

```
8*symsum((r3^2*exp(r3*t))/(6*(8*r3^2 + 10*r3 + 3)), r3 in RootOf(s3^3
+ (15*s3^2)/8 + (9*s3)/8 + 3/16, s3))
```

Note that pretty is not used here because it's harder to read. The answer is a symbolic sum over the three roots of the cubic equation in s3. The different behaviour occurs because, in this example, the roots of the denominator polynomial are long expressions involving cube roots, and the result is complicated. You can check this with

```
>> syms s3
>> solve(s3^3 + (15*s3^2)/8 + (9*s3)/8 + 3/16, s3)
```

(Note that the "=0" is assumed.)

Problems with f in this form are that an error occurs if we attempt to plot it using ezplot. To get around these problems, we can use **variable precision arithmetic** with the vpa function, and replace numbers in f by approximations to a specified number of decimal digits. Thus

```
>>f1 = vpa(f, 5);
>>f2 = simple(f1);
>>f3 = vpa(f2,5)
f3 =
0.1136*exp(-0.27229*t) + 0.3864*exp(-0.80135*t)*cos(0.21547*t) -
2.7704*exp(-0.80135*t)*sin(0.21547*t)
```

and f3 can be plotted with ezplot. Note that simple will give a warning suggesting you use simplify, but if you do, you may get an answer without the sin and cos simplification.

Connection between Laplace Transforms and circuit analysis

In the analysis of circuits with sources which vary in an arbitrary manner with time, it is necessary to solve differential equations, either directly or indirectly. The Laplace transform (DS Chapter 14) is a powerful tool which allows analysis to be carried out

in the complex frequency domain (in terms of the complex frequency *s*) rather than the time domain (in terms of time *t*).

Voltages $v(t)$ and currents $i(t)$ are replaced by their Laplace transforms $V(s)$ and $I(s)$, and the impedances of circuit elements are expressed in terms of *s* as follows:

a resistance of *R* ohms has an impedance of *R* ohms;

an inductance of *L* Henries has an impedance of *sL* ohms; and

a capacitance of *C* Farads has an impedance of 1/(*sC*) ohms.

Rules for combining impedances in series, etc, are as for resistors in dc analysis, and many of the techniques of dc analysis can be used.

Relationship between Laplace analysis, dc analysis and complex phasor analysis

Suppose we have a circuit consisting of R, L and C elements and we have obtained general results in terms of Laplace transforms. Then

Results for steady state dc conditions can be obtained by taking $s \to 0$, and

Results for steady state sinusoidal conditions can be obtained by substituting $s = j\omega$ where ω is the frequency in rad/s.

Exercises

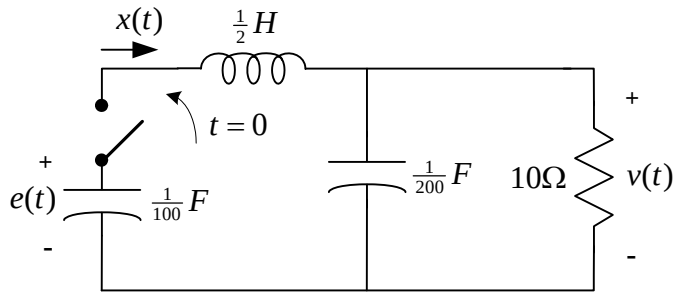
Exercise 1

Use dsolve to find solutions for $f(x)$ and $g(x)$ if

$$\frac{df(x)}{dx} + 1.5f(x) - g(x) = 0$$
$$\frac{dg(x)}{dx} + f(x) - 2g(x) = 0$$

with $f(0) = 0,\ g(0) = 2$. You may find it useful to use vpa and simplify to provide a compact display on the screen. Note that the current Matlab versions of simple and simplify may give odd looking results.

Exercise 2

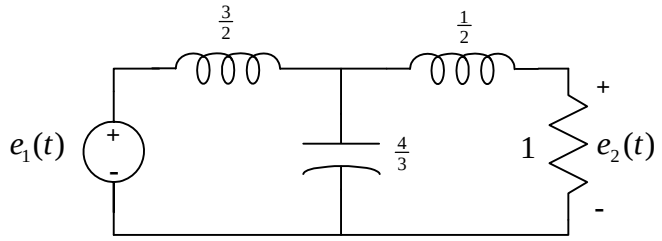


In the circuit shown, the 1/100 F capacitor is charged to 5 V with the switch open and there is no charge on the 1/200 F capacitor. At *t* = 0, the switch is closed. Write down three differential equations in terms of the voltages $e(t)$, $v(t)$ and the current $x(t)$. Use

MATLAB's dsolve function to find a solution for $e(t),\ t > 0$ and plot the result over suitable ranges. Use vpa and pretty to display a compact readable solution on the screen.

Exercise 3

In the circuit shown, component values are in Ω, F and H.



Use Laplace transform methods to determine the step response of the network shown, i.e., if $e_1(t)$ is the unit step function ($e_1(t) = 0,\ t < 0$ and $e_1(t) = 1,\ t > 0$), how does the output voltage $e_2(t)$ behave for $t \geq 0$? Such an excitation could be caused by a dc voltage source switched into the circuit at $t = 0$. Assume that, initially, the capacitor is uncharged and the inductors carry no current. Use the method of mesh currents (DS 4.6). Steps in solving the problem are:

(a) Redraw the circuit, showing all elements in the complex impedance form (in terms of s), and show the voltages as their Laplace transforms.

(b) On the same diagram, show the loop currents (Laplace transform form) as $I_1(s), I_2(s)$, say.

(c) Write the equations required to solve for $E_2(s), I_1(s), I_2(s)$ in terms of $E_1(s)$.

(d) Write a script M-file that solves these equations, substitutes the Laplace transform of $E_1(s)$ to find the Laplace transform of $E_2(s)$, finds $e_2(t)$ by taking the inverse transform and plots $e_2(t)$ for a suitable range of positive *t*.

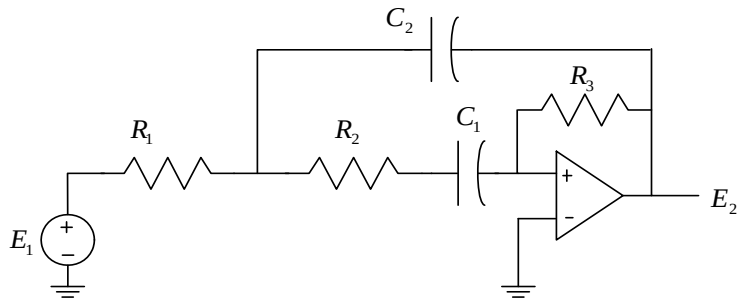
Exercise 4

In Exercise 3, the ratio $G(s) = E_2(s)/E_1(s)$ of Laplace transform of output to Laplace transform of input is the *voltage gain*. (In the more general setting of linear systems analysis, such functions are called *transfer functions*). In the special case when $s = j\omega$, $G(j\omega)$ is referred to as the *frequency response*.

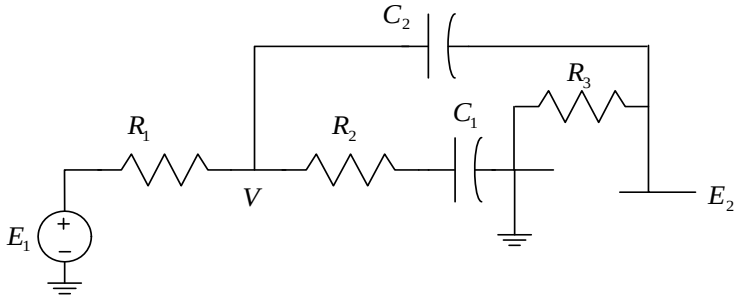
Rewrite the M-file of Exercise 3 to calculate and plot the frequency response of the network, i.e., plot $|G(j\omega)|$ versus ω. This circuit, in fact, acts as a low pass filter. Do your results support this description?

Exercise 5

An important class of electronic circuits involves operational amplifiers. Consider the problem, in the circuit shown, of determining the voltage gain $E_2(s)/E_1(s)$. We will assume that the op amp has (i) infinite input impedance and (ii) infinite voltage gain. These are often very reasonable assumptions.



Because of the first assumption, no current flows into the + input terminal. Because of the second, since E_2 is finite, the differential input voltage is zero and the + input is at a "virtual ground" potential. Thus, to determine the voltages and currents, we need to analyse the following circuit:



Find the voltage gain. As good a way as any is to use Kirchhoff's laws to write and solve two equations in V and E_2.

Exercise 6: Random Numbers

Computer simulations frequently require the use of random numbers. The idea of using a deterministic machine to produce random numbers seems to be an oxymoron. Indeed, modern computers generate *pseudo-random* numbers. These pseudo-random numbers pass a series of statistical tests that indicate the numbers "appear" random. In the early days of computers, John von Neumann suggested the middle-square method for generating random numbers. This method starts with an initial four-digit integer called the seed, squares it to get an eight-digit number, and then takes the middle four digits as the next random number. This will, of course, only produce random numbers between 0 and 9999. MATLAB's random number generator, on the other hand, can theoretically produce 2^{1492} different random numbers before repeating itself (the 1492 has led some to call this random number generator the "Christopher Columbus" generator). Amazingly, this generator uses shift register and bit manipulations that require no multiplication or division operations. Even if we were to consume random numbers at a rate of one million per second, it would take more than 10^{435} years (about 10^{425} times the age of the universe) before the sequence would repeat. (Note that this is only true for MATLAB v.5 and greater.)

Uniform random numbers between 0 and 1 and also matrices of random numbers are generated in MATLAB using the **rand** function:

```
>> rand(10,10).
```

The command ‘randn’ produces a Gaussian distribution of random numbers with a mean of zero and variance of one:

>> randn(10,10).

Unless the initial seed is changed, MATLAB’s random number generator produces the same random sequence of numbers for every execution. Therefore, the following command can be used to change the initial seed:

>> rng(100*sum(clock)).

This command uses the software clock maintained by MATLAB to compute an initial seed for the generator. The variable clock is a six-element vector with the format of [year month day hour minute second]. Try typing ‘**clock**’ at the MATLAB prompt. The command ‘sum’ adds the six elements of the vector and the multiplication by 100 results in a final integer to seed the random number generator.

Calculating pi

One way to calculate π is to randomly and uniformly pick points in the unit square with vertices (0,0), (1,0), (0,1), (1,1) and calculate the proportion of points which lie within unit distance of the origin. This number will be proportional to the value of π. Why? Write a MATLAB function to calculate the value of π using this method. Let the number of points to pick be an input to your function. This program can be written with or without a for loop. Note that this method is a very inefficient way to calculate the value of π. Nonetheless, the procedure outlined here is the “essence” of the Monte Carlo techniques found in PSPICE that are used in simulating the influence of the variations of design parameters on integrated circuits.