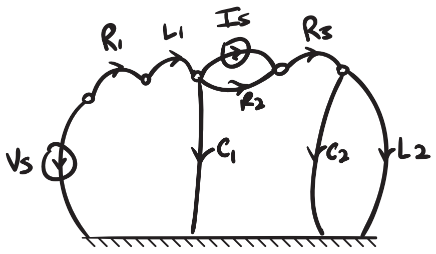
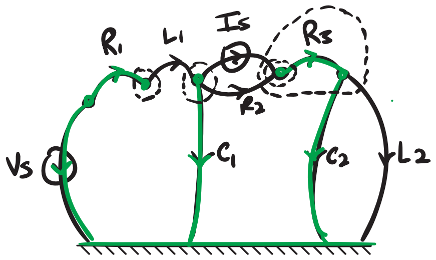
imp.examat Impedance modeling example in Matlab

Example imp.examat–1 re: Impedance modeling with Matlab

1 Consider the linear graph of an electronic system, below. Use impedance methods to derive the transfer functions from inputs V_S and I_S to outputs v_{C_2} and i_{R_1}.



2 The normal tree is shown, below, along with contours to be used for continuity equations.



3 We switch over to Matlab for the remainder of the solution.
Let's define the required symbolic variables.

```
syms s VS IS ...
  vC1 iC1 vC2 iC2 vL1 iL1 vL2 iL2 ...
  vR1 iR1 vR2 iR2 vR3 iR3 ...
  zC1 zC2 zL1 zL2 zR1 zR2 zR3 ...
  C1 C2 L1 L2 R1 R2 R3
```

We also specify the unknown variables (two for each passive element), output variables, and input variables.

```
unknowns = [...
  vC1 iC1 vC2 iC2 vL1 iL1 vL2 iL2 ...
  vR1 iR1 vR2 iR2 vR3 iR3 ...
];
out_i = [3,10]; % output indices
in = [VS;IS]; % input variables
```

Now let's define our elemental, continuity, and compatibility equations.

```
elemental = [...
  vC1 == iC1*zC1,...
  vC2 == iC2*zC2,...
  vL1 == iL1*zL1,...
  vL2 == iL2*zL2,...
  vR1 == iR1*zR1,...
  vR2 == iR2*zR2,...
  vR3 == iR3*zR3 ...
];
continuity = [...
  iC1 == iL1 - IS - iR2,...
  iR1 == iL1,...
  iC2 == IS + iR2,...
  iR3 == IS + iR2 ...
];
compatibility = [
  vL1 == -vR1 + VS - vC1,...
  vL2 == vC2,...
  vR2 == vC1 - vC2 - vR3...
];
```

These form a linear system of $2 \times 7 = 14$ unknowns and 14 equations. Such systems can be defined in matrix form as `M*unknowns == b`, where `M` are the coefficients of the unknowns, `unknowns` is the vector of unknowns, and `b` is the vector of terms that include the inputs. Matlab has the function `equationsToMatrix` for specifying the matrix form from a list of equations.

```
[M,b] = equationsToMatrix(...
  [elemental,continuity,compatibility],... % eq's
  unknowns... % unknown variables
);
disp('first 10 columns of M:') % to fit on screen
disp(M(:,1:10))
disp('b transposed:') % for pretty
disp(b.')
```

```
first 10 columns of M:
[  1, -zC1,  0,    0, 0,    0, 0,    0, 0,    0]
[  0,    0,  1, -zC2, 0,    0, 0,    0, 0,    0]
[  0,    0,  0,    0, 1, -zL1, 0,    0, 0,    0]
[  0,    0,  0,    0, 0,    0, 1, -zL2, 0,    0]
[  0,    0,  0,    0, 0,    0, 0,    0, 1, -zR1]
[  0,    0,  0,    0, 0,    0, 0,    0, 0,    0]
[  0,    0,  0,    0, 0,    0, 0,    0, 0,    0]
[  0,    1,  0,    0, 0,   -1, 0,    0, 0,    0]
[  0,    0,  0,    0, 0,   -1, 0,    0, 0,    1]
[  0,    0,  0,    1, 0,    0, 0,    0, 0,    0]
[  0,    0,  0,    0, 0,    0, 0,    0, 0,    0]
[  1,    0,  0,    0, 1,    0, 0,    0, 1,    0]
[  0,    0, -1,    0, 0,    0, 1,    0, 0,    0]
[ -1,    0,  1,    0, 0,    0, 0,    0, 0,    0]

b transposed:
[ 0, 0, 0, 0, 0, 0, 0, -IS, 0, IS, IS, VS, 0, 0]
```

Furthermore, Matlab has the function `linsolve` for solving for the `unknowns`.

```
sol_z = linsolve(M,b);
```

This solution `sol_z` includes impedances. We would like to substitute for the actual impedance values, defined as follows in `struct` form.

```
impedances.zC1 = 1/(C1*s);
impedances.zC2 = 1/(C2*s);
impedances.zL1 = L1*s;
impedances.zL2 = L2*s;
impedances.zR1 = R1;
impedances.zR2 = R2;
impedances.zR3 = R3;
```

Now we can substitute impedances with `subs`, which gives us a solution in terms of `s`.

```
sol = simplify(...
  subs(...
    sol_z,...
    fieldnames(impedances),...
    struct2cell(impedances)...
  )...
);
[n,d]=numden(sol);
sol_nd = collect([n,d],s);
```

Finally, we can compute the transfer function matrix H(s) by using the solutions for our outputs and substituting 1 for the input of interest and 0 for the others (this code generalizes to more than two inputs).

```
for input_i = 1:length(in) % each input
  for output_i = 1:length(out_i) % each output
    output_index = out_i(output_i); % idx of ops var
    input_var = in(input_i); % input variable
    other_inputs = setdiff(in,[input_var]); % sneaky
    num = sol_nd(output_index,1); % w/all ip's
    den = sol_nd(output_index,2); % w/all ip's
    num_tf = subs(... % eliminate other inputs
      num,...
      [input_var,other_inputs],...
      [1,zeros(size(other_inputs))]...
    );
    den_tf = subs(... % eliminate other inputs
      den,...
      [input_var,other_inputs],...
      [1,zeros(size(other_inputs))]...
    );
    H(input_i,output_i) = ... % collect s and divide
      collect(num_tf,s)/collect(den_tf,s);
  end
end
pretty(H(2,1)) % display H_21
```

```
            2
(C1 L1 R2 s  + C1 R1 R2 s + R2)/((C1 C2 L1 R2 + C1
↪   C2 L1 R3)

    3
   s  + (C1 L1 + C2 L1 + C1 C2 R1 R2 + C1 C2 R1 R3)

    2
   s  + (C1 R1 + C2 R1 + C2 R2 + C2 R3) s + 1)
```