Time Domain Simulation of Analog/RF Circuits for MATLAB / SIMULINK / C++

James A Crawford

Synopsis[i]

RF systems-level designers frequently have a need to incorporate continuous-time elements like RF filters and phase-locked loop related analog circuitry into their simulations. These simulations are frequently done in MATLAB, SIMULINK, or C/C++. Compiled solutions are most attractive for time-intensive analyses efforts (e.g., Monte-Carlo technique).

Many designers do not have access to higher level design tools (e.g., ADS, Cadence) which provide convenient behavioral models for such blocks. In such cases, the designer must frequently come up with their own set of state-equations which are then solved as a set of simultaneous differential (or difference) equations. Formulating these equations is both tedious and prone to error.

This paper presents a convenient means to assemble the underlying discrete-time solution to such problems based upon straight-forward matrix techniques. This formulation can be implemented in the designer's analysis tool of choice.

1 Problem Statement

The simple circuit example shown in Figure 1 will be used to illustrate the underlying problem being addressed. The voltage transfer function for this simple network is given by[1]

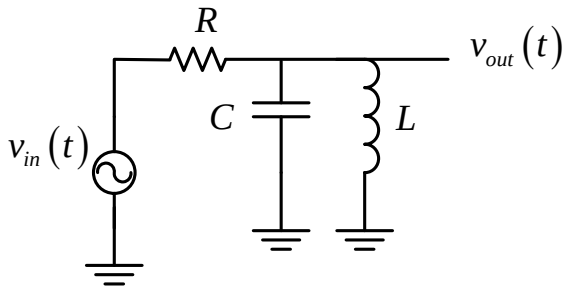


Figure 1 RLC example circuit

$$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{s/\tau}{s^2 + s/\tau + \omega_n^2} \tag{1}$$

where

$$\tau = RC$$
$$\omega_n = \frac{1}{\sqrt{LC}} \tag{2}$$

This transfer function is normally written in its standard form as

$$H(s) = \frac{2\varsigma\omega_n s}{s^2 + 2\varsigma\omega_n s + \omega_n^2} \tag{3}$$

with $2\varsigma\omega_n = 1/(RC)$. The associated time-domain unit-step response is given by

$$v(t) = \frac{2\varsigma}{\sqrt{1-\varsigma^2}} \exp(-\varsigma\omega_n t) \sin\left(\omega_n \sqrt{1-\varsigma^2}\, t\right) \quad for \quad t \geq 0 \tag{4}$$

With the additional constraint $\varsigma < 1$. All too often, engineers numerically approximate time derivatives of a state variable *u* as

$$\frac{du}{dt} \cong \frac{u(t+h) - u(t)}{h} \tag{5}$$

[1] From §8.2 of [1].

using a relatively small time increment *h* and then formulate a discrete-time solution based upon this approximation. While appearing quite simple, this method (known as the forward Euler method[2]) is poorly conditioned from a stability standpoint and is also rather imprecise.

Many excellent references are available which address numerical integration, but the primary emphasis in this paper is the incorporation of the time-domain solution into a simplified matrix solution. In pursuing this objective, three well-known numerical integration algorithms will first be introduced. Each technique will then be incorporated into a matrix-based solution of Figure 1 which can be easily modified for arbitrarily large circuit applications.

The state-equations associated with (3) can be written as

$$\begin{aligned} \frac{du_1}{dt} &= -2\zeta\omega_n u_1 - \omega_n^2 u_2 + 2\zeta\omega_n \\ \frac{du_2}{dt} &= u_1 \end{aligned} \tag{6}$$

This pair of equations is not difficult to formulate whereas the formulation for a more complicated schematic like Figure 2 would obviously be much more involved.

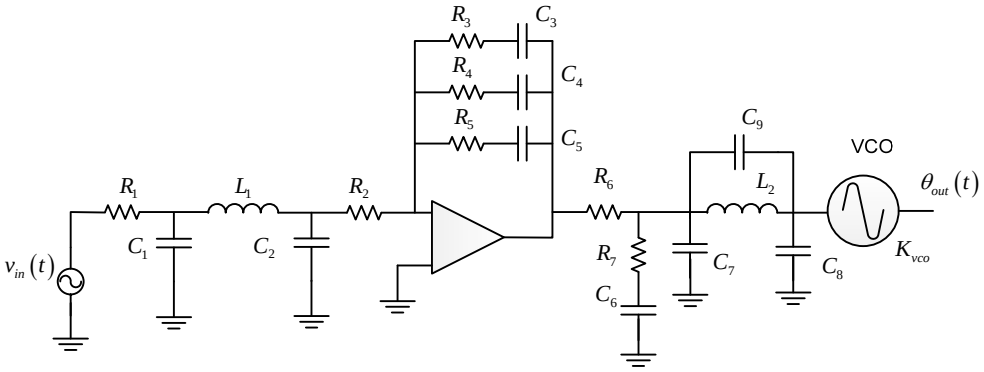


Figure 2 Second circuit example with much greater complexity than Figure 1

2 Numerical Integration Methods

The four numerical integration methods considered in this paper are

- Forward Euler
- Backward Euler
- Trapezoidal
- 2nd-Order Gear

2.1 Forward Euler

In the forward Euler case, time derivatives are approximated as given by (5). More explicitly for the integration case,

[2] Also known as the first-order Adams-Bashforth method.

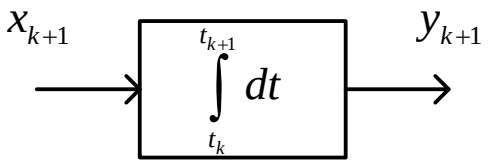


Figure 3 Simplistic representation for a discrete-time integration block where $t_{k+1} = t_k + h$

$$y_{k+1} = y_k + hx_k \tag{7}$$

This simple form is used fairly frequently as an ad-hoc method. Other methods are much better as developed herein.

2.2 Algorithm Stability

In the case of a multistep numerical integration algorithm, numerical stability can be assessed by considering the first-order initial value problem given by

$$\frac{dy}{dt} = -\lambda y \tag{8}$$

where y(0) = 1 is assumed. In terms of discrete-time samples, (8) is given by

$$\dot{y}_k = -\lambda y_k \tag{9}$$

Substituting (9) into (7) where by definition $\dot{y}_k \equiv x_k$ results in

$$\begin{aligned} y_{k+1} &= y_k + hx_k \\ &= y_k + h\left(-\lambda y_k\right) \end{aligned} \tag{10}$$

where *h* is a suitably small time increment. This is easily reorganized as

$$y_{k+1} = y_k\left(1 - h\lambda\right) \tag{11}$$

This result can be rewritten as

$$\frac{y_{k+1}}{y_k} = 1 - h\lambda \tag{12}$$

from which it is easy to conclude that the solution geometrically increases without bound of $\left|1 - h\lambda\right| > 1$. Denoting $h\lambda = u + jv$, the solution consequently diverges for

$$(1-u)^2+v^2>1 \tag{13}$$

which corresponds to the exterior of a unit-circle centered at the point (1,0) as shown in Figure 4. The numerical precision of this method is compared to the other methods later.

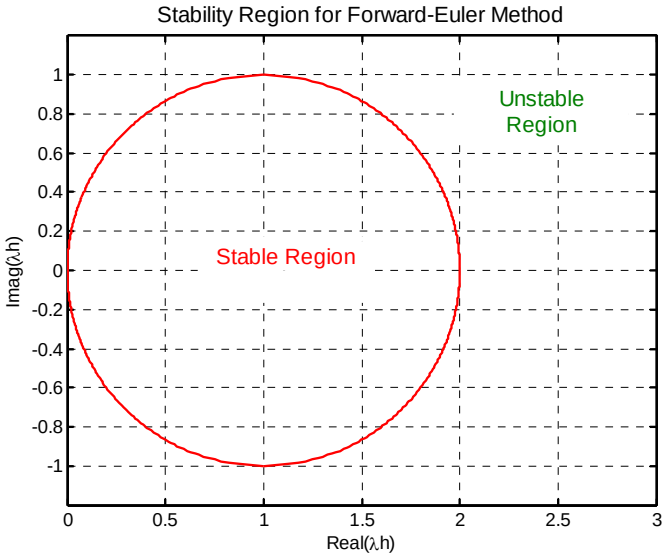


Figure 4 Stability regions for forward Euler method[ii]

In the more systematic stability analysis approach is needed to address the general case. A multistep numerical integration algorithm can be written as

$$y_{k+1}=\sum_{i=0}^{p} a_i y_{k-i}+h\sum_{i=-1}^{p} b_i \dot{y}\left(y_{k-i},t_{k-i}\right) \tag{14}$$

The multistep algorithm is said to be absolutely stable[3] for those values of $h\lambda$ for which the $p+1$ roots of the characteristic equation lie within or on the unit circle $|z|=1$. Returning to (11), the difference in time sample index is handled by making use of the unit time-delay operator z^{-1} which translates (11) into

$$\begin{aligned} yz &= y\left(1-h\lambda\right) \\ y\left(z-1+h\lambda\right) &= 0 \end{aligned} \tag{15}$$

The characteristic equation is given by the quantity within the parenthesis as

$$P\left(z\right)=z-1+h\lambda \tag{16}$$

The single root of $P\left(z\right)$ is given by

[3] From §13.1 of [2].

$$z = 1 - h\lambda \tag{17}$$

and so long as this root lies within the unit circle, the forward Euler formula will be stable. This *characteristic function* method is used to assess stability for the trapezoidal and 2nd-order Gear methods as well.

2.3 Backward Euler Method

The backward Euler method is almost identical to the forward Euler method (7) except for a difference in the time index. This small difference is extremely important, however, thereby creating an implicit integration formula rather than an explicit formula like the forward Euler method. This integration formula is given by

$$y_{k+1} = y_k + hx_{k+1} \tag{18}$$

Algorithm stability in the context of the first-order differential equation given (8) transforms (18) into

$$y_{k+1} = y_k + h\left(-\lambda y_{k+1}\right) \tag{19}$$

from which it follows

$$\frac{y_{k+1}}{y_k} = \frac{1}{1+h\lambda} \tag{20}$$

This result shows that the method is stable so long as $\left|1+h\lambda\right| > 1$ which corresponds to the exterior of the unit-circle shown in Figure 5.

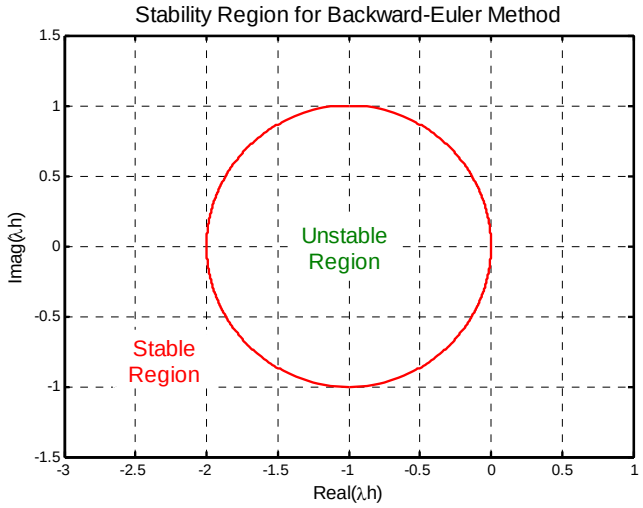


Figure 5 Stability regions for backward Euler method[iii]

2.4 Trapezoidal Method

The trapezoidal integration method for a discrete-sample input x_k is given by

$$y_{k+1} = y_k + \frac{x_{k+1} + x_k}{2} h \tag{21}$$

Denoting a unit-sample time-delay *h* by z^{-1}, (21) can be rewritten as

$$\frac{y_{k+1}}{x_{k+1}} = \left[\frac{1+z^{-1}}{1-z^{-1}}\right]\frac{h}{2} \tag{22}$$

In the Laplace transform domain, $y(s)/x(s) = 1/s$ since the purpose of (21) is to mimic an ideal integrator as far as possible. Based upon this and (22), the trapezoidal method approximates the Heaviside operator *s* as

$$s \approx \frac{2}{h}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \tag{23}$$

This is none other than the well-known bi-linear transform relationship that frequently occurs in digital signal processing.

The stability of this method can be assessed in the same way as done previously for the forward and backward Euler methods. From (8),

$$\dot{y}_k = -\lambda y_k = x_k \tag{24}$$

and using this in (21) produces

$$y_{k+1} = y_k - \left(\frac{h\lambda}{2}\right)\left(y_{k+1} + y_k\right) \tag{25}$$

resulting in

$$\frac{y_{k+1}}{y_k} = \frac{1-\frac{h\lambda}{2}}{1+\frac{h\lambda}{2}} \tag{26}$$

The magnitude of the right-hand side of (26) must be < 1 in order to have a stable result. This condition is satisfied so long as $\mathrm{Re}(h\lambda) > 0$ which corresponds to the entire right half-plan of Figure 4. The stability region for trapezoidal integration is consequently much larger than for the forward Euler method.

Looking at the trapezoidal method's stability by way of the characteristic function approach, (21) is the starting point. Making use of the unit time-delay operator in (21) along with (24) results in

$$y\left(z - 1 + \frac{h\lambda}{2}z + \frac{h\lambda}{2}\right) = 0 \tag{27}$$

and the solution to the characteristic equation is given by

$$z = \frac{\frac{h\lambda}{2} - 1}{\frac{h\lambda}{2} + 1} \tag{28}$$

which gives the same stability / instability regions as found above.

2.5 2nd-Order Gear

The 2nd-order Gear numerical integration formula is given by

$$y_{k+1} = a_0 y_k + a_1 y_{k-1} + h b_{-1} \dot{y}_{k+1} \tag{29}$$

where $a_0 = 4/3$, $a_1 = -1/3$, and $b_{-1} = 2/3$. In considering the stability of this method in the context of (8), $\dot{y}_{k+1} = -\lambda y_{k+1}$ which transforms (29) into

$$y_{k+1} = a_0 y_k + a_1 y_{k-1} - h b_{-1} \lambda y_{k+1} \tag{30}$$

Using z^{-1} to represent a unit time-delay once more transforms (30) into

$$\begin{aligned} & y(z) = a_0 z^{-1} y(z) + a_1 z^{-2} y(z) - h b_{-1} \lambda y(z) \\ & \Rightarrow y(z)\left[1 + h b_{-1}\lambda - a_0 z^{-1} - a_1 z^{-2}\right] = 0 \\ & \Rightarrow y(z)\left[1 + \frac{2}{3}h\lambda - \frac{4}{3}z^{-1} + \frac{1}{3}z^{-1}\right] \end{aligned} \tag{31}$$

The integration formula will be unstable for values of $h\lambda$ for which the bracketed quantity is zero for *z* anywhere on the unit-circle, or equivalently for

$$2e^{j\theta} - \frac{1}{2}e^{j2\theta} - \frac{3}{2} = h\lambda \tag{32}$$

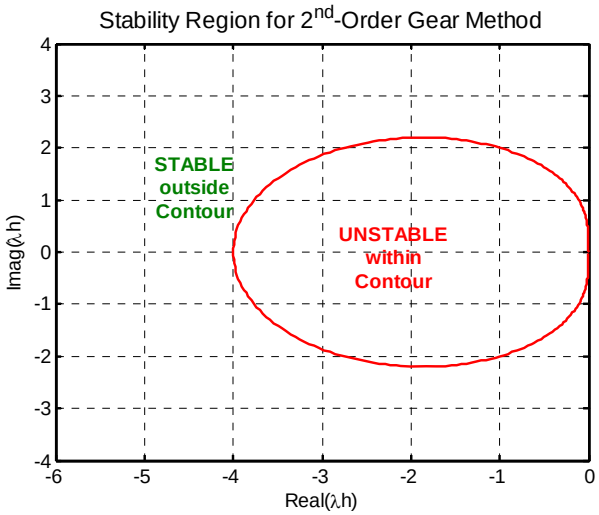


Figure 6 Stability region for 2nd-order Gear method[iv]

3 Numerical Precision

The starting point for this discussion is the set of two simultaneous differential equations given earlier (6) which are repeated here for convenience.

$$\begin{aligned} \frac{du_1}{dt} &= -2\varsigma\omega_n u_1 - \omega_n^2 u_2 + 2\varsigma\omega_n \\ \frac{du_2}{dt} &= u_1 \end{aligned} \tag{33}$$

3.1 Forward Euler Case

The associated integration formula was given earlier for this case as (7). This equation can be arranged into the differential equation form as

$$\dot{u}_k = \frac{u_{k+1} - u_k}{h} \tag{34}$$

Using this form in (33) produces

$$\begin{aligned}\frac{u_{1_{k+1}} - u_{1_k}}{h} &= -2\varsigma\omega_n u_{1_k} - \omega_n^2 u_{2_k} + 2\varsigma\omega_n \\ \frac{u_{2_{k+1}} - u_{2_k}}{h} &= u_{1_k}\end{aligned} \tag{35}$$

Straight forward algebraic arrangement results in the convenient form

$$\begin{bmatrix} u_{1_{k+1}} \\ u_{2_{k+1}} \end{bmatrix} = \begin{bmatrix} (1 - 2\varsigma\omega_n h) & -\omega_n^2 h \\ h & 1 \end{bmatrix} \begin{bmatrix} u_{1_k} \\ u_{2_k} \end{bmatrix} + \begin{bmatrix} 2\varsigma\omega_n h \\ 0 \end{bmatrix} \tag{36}$$

3.2 Backward Euler Case

This numerical method has not been mentioned until now, but will serve useful for later comparisons. This integration formula is given by

$$y_{k+1} = y_k + hx_{k+1} \tag{37}$$

The discrete-time solution to (33) using this integration formula is

$$\begin{bmatrix} u_{1_{k+1}} \\ u_{2_{k+1}} \end{bmatrix} = \frac{\begin{bmatrix} \frac{1}{h} & -\omega_n^2 \\ 1 & \left(\frac{1}{h} + 2\varsigma\omega_n\right) \end{bmatrix} \begin{bmatrix} \frac{u_{1_k}}{h} + 2\varsigma\omega_n \\ \frac{u_{2_k}}{h} \end{bmatrix}}{\frac{1}{h}\left(\frac{1}{h} + 2\varsigma\omega_n\right) + \omega_n^2} \tag{38}$$

This result can be iterated to compute the impulse response represented by (33) using the backward Euler method.

3.3 Trapezoidal Case

The associated integration formula was given earlier for this case as (21). This can be rearranged as

$$\frac{2}{h}\left(u_{k+1}-u_k\right)=\dot{u}_{k+1}+\dot{u}_k \tag{39}$$

Using this formula along with (33) can be used to create the discrete-time solution

$$\begin{bmatrix} u_{1_{k+1}} \\ u_{2_{k+1}} \end{bmatrix}=\frac{\begin{bmatrix} \frac{2}{h} & -\omega_n^2 \\ 1 & \left(\frac{2}{h}+2\varsigma\omega_n\right) \end{bmatrix}\begin{bmatrix} \frac{2}{h}u_{1_k}-2\varsigma\omega_n u_{1_k}-\omega_n^2 u_{2_k}+4\varsigma\omega_n \\ \frac{2}{h}u_{2_k}+u_{1_k} \end{bmatrix}}{\frac{2}{h}\left(\frac{2}{h}+2\varsigma\omega_n\right)+\omega_n^2} \tag{40}$$

This result can be iterated to compute the impulse response represented by (33) using the trapezoidal method.

3.4 2nd Order Gear Case

The associated integration formula was given earlier for this case as (29). This can be arranged as

$$\dot{u}_{k+1}=\frac{3}{2h}\left[u_{k+1}-\frac{4}{3}u_k+\frac{1}{3}u_{k-1}\right] \tag{41}$$

The resultant discrete-time solution is

$$\begin{bmatrix} u_{1_{k+1}} \\ u_{2_{k+1}} \end{bmatrix}=\frac{\begin{bmatrix} \frac{3}{2h} & -\omega_n^2 \\ 1 & \left(\frac{3}{2h}+2\varsigma\omega_n\right) \end{bmatrix}\begin{bmatrix} \frac{2}{h}u_{1_k}-\frac{1}{2h}u_{1_{k-1}}+2\varsigma\omega_n \\ \frac{2}{h}u_{2_k}-\frac{1}{2h}u_{2_{k-1}} \end{bmatrix}}{\left(\frac{3}{2h}\right)\left(\frac{3}{2h}+2\varsigma\omega_n\right)+\omega_n^2} \tag{42}$$

3.5 Comparison of Results

The computed impulse responses for several different time-steps are shown in Figure 7 through Figure 9. The nearly unstable behavior of the forward Euler method for the coarse time-step used in Figure 7 should serve as a good reminder to avoid using this integration method in general. It is admittedly a bit unfair to compare the first-order methods (forward & backward Euler) with the second-order methods (trapezoidal & 2nd-order Gear) but this too is done to emphasize why the 2nd order methods are to be preferred.

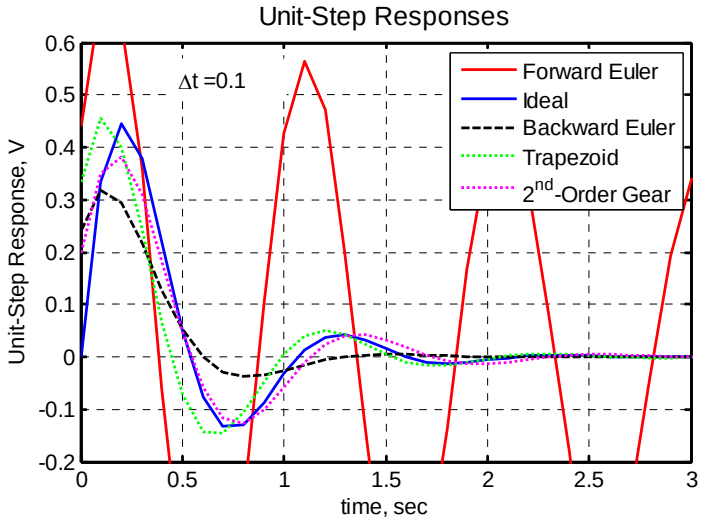


Figure 7 Comparison of results for a very coarse time-step Δt = 0.10 sec

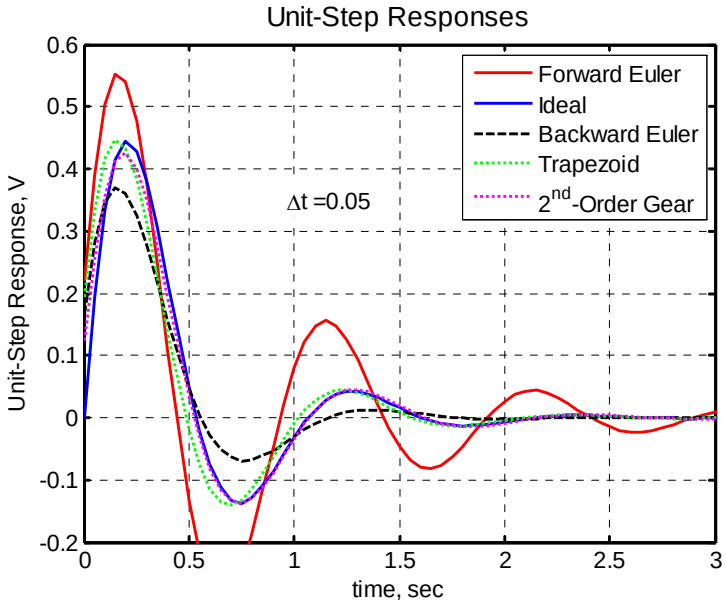


Figure 8 Comparison of results for a time-step Δt = 0.05 sec

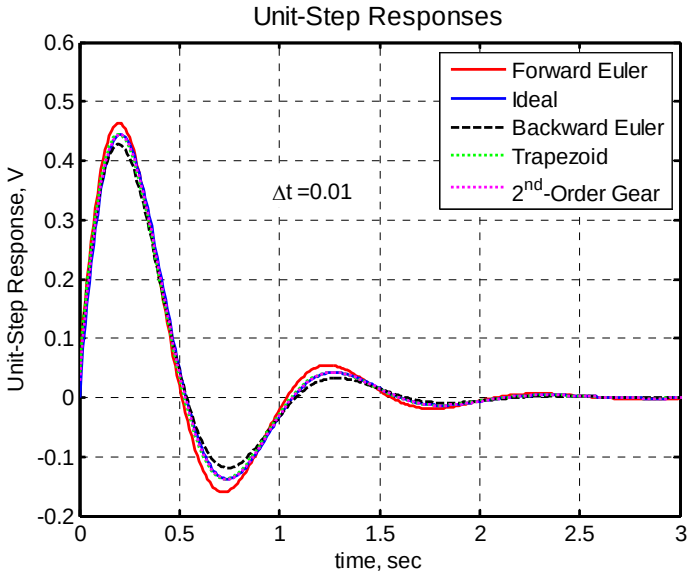


Figure 9 Comparison of results for a time-step $\Delta t = 0.01$ sec

It is worthwhile to compare the results relative to the ideal response given by (4). In general, numerical integration algorithms introduce a small time delay compared to the ideal response. Normally this delay is a true time-delay but in some cases it may be a small time advance as well. The trapezoidal and 2nd-order Gear results are compared to a time-advanced version of the ideal response in Figure 10, a time-delayed version in Figure 11, and no time adjustment to the ideal response in Figure 12. At least for this particular impulse response and Δt value, the 2nd-order Gear appears to be shape-preserving as evidenced by Figure 10.

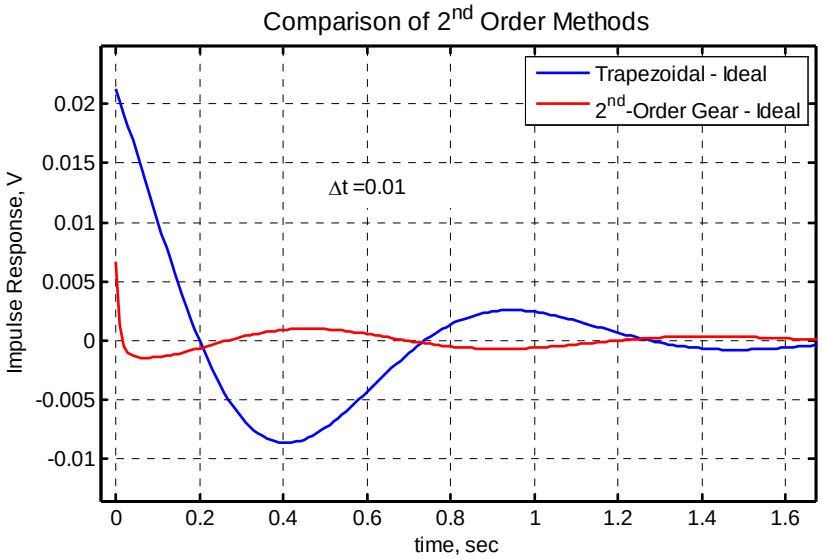


Figure 10 Comparison of results with respect to *ideal* where the ideal result has been advanced in time by $\Delta t/2$

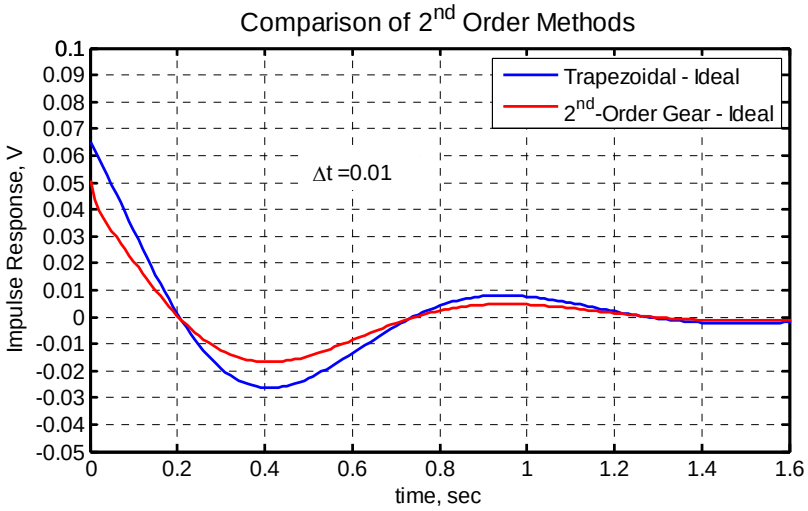


Figure 11 Comparison of results with respect to *ideal* where the ideal result has been delayed in time by $\Delta t/2$

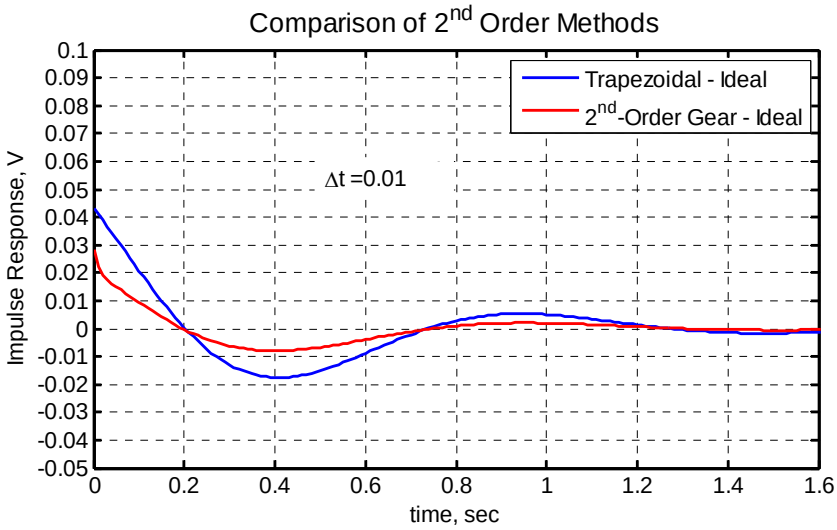


Figure 12 Comparison of results with respect to *ideal* where the ideal result has neither been delayed or advanced in time

The trapezoidal and 2nd-order Gear methods are both desirable methods to use. The specific choice depends in part on how easy the discrete-time equations are formulated for actual circuits. This is the topic of the next section.

In small over-sampling situations (e.g., sampling rate < 20x the maximum frequency present in the circuit), the 2nd-order Gear method should give better results than the trapezoidal method. For higher over-sampling ratios, both methods will perform quite well.

4 Discrete Time Circuit Equivalents

This section develops the implementation details for discrete-time simulation of analog circuits using the preferred trapezoidal and 2nd-order Gear methods. Each of these methods will be developed separately.

4.1 Trapezoidal Method

In the case of an ideal capacitor, the defining relationship is

$$i = C\frac{dV}{dt} \tag{43}$$

where the polarities are as shown in Figure 13. The trapezoidal integration method represents a time derivative as given by (39). Combining these two equations produces

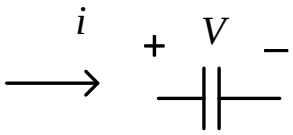


Figure 13 Basic capacitor with current direction and voltage polarity shown

$$\begin{aligned} i_{k+1} + i_k &= C\frac{dV_{k+1}}{dt} + C\frac{dV_k}{dt} = \frac{2C}{h}\left(V_{k+1} - V_k\right) \\ \therefore i_{k+1} + i_k &= \frac{2C}{h}\left(V_{k+1} - V_k\right) \end{aligned} \tag{44}$$

This result can be re-cast as an equivalent circuit as shown in Figure 14.

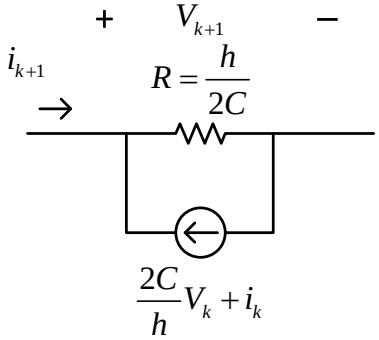


Figure 14 Companion model for ideal capacitor based upon the trapezoidal numerical integration formula

The same type of formulation can be created for an ideal linear inductor. Starting with the voltage and current polarities shown in Figure 15 and

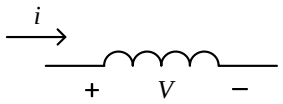


Figure 15 Ideal linear inductor

the fundamental relationship for inductors given by

$$v = L\frac{di}{dt} \tag{45}$$

the trapezoidal integration formula (39) can be combined as

$$\begin{aligned} &L\left(\frac{di_{k+1}}{dt}+\frac{di_k}{dt}\right)=\frac{2L}{h}\left(i_{k+1}-i_k\right)=V_{k+1}+V_k \\ &\therefore \frac{h}{2L}\left(V_{k+1}+V_k\right)-i_{k+1}+i_k=0 \end{aligned} \tag{46}$$

with the equivalent circuit representation shown in Figure 16.

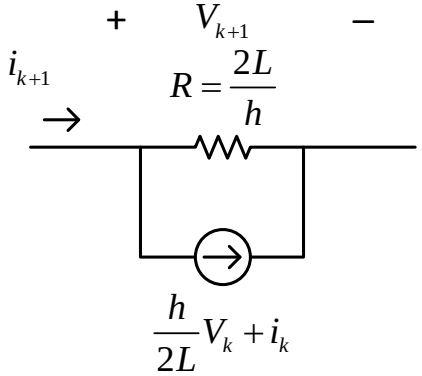


Figure 16 Companion model for ideal inductor based upon the trapezoidal numerical integration formula

These results may look somewhat complicated, but they permit an arbitrary circuit of resistors, capacitors, and inductors to be represented in the time domain as a *fixed* array of conductances driven by current sources thereby making matrix-based iterative computations very straight forward as will be shown shortly.

4.2 2nd-Order Gear Method

The discrete-time formula associated with an ideal capacitor is given b

$$\frac{C}{hb_{-1}}\left(V_{k+1}-a_0V_k-a_1V_{k-1}\right)=C\left.\frac{dV}{dt}\right|_{k+1}=i_{k+1} \tag{47}$$

This result can be cast into the equivalent circuit model shown here in Figure 17 where the proper substitutions for the Gear coefficients (a_0, a_1, b_{-1}) have been made.

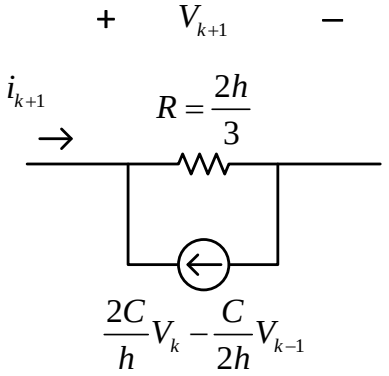


Figure 17 2nd-order Gear companion circuit model for an ideal capacitor C

For an ideal inductor, the 2nd-order Gear integration formula results in

$$L\frac{di}{dt}\Rightarrow\frac{L}{hb_{-1}}\left(i_{k+1}-a_0i_k-a_1i_{k-1}\right)=V_{k+1} \tag{48}$$

This result can be cast into the equivalent circuit model shown here in

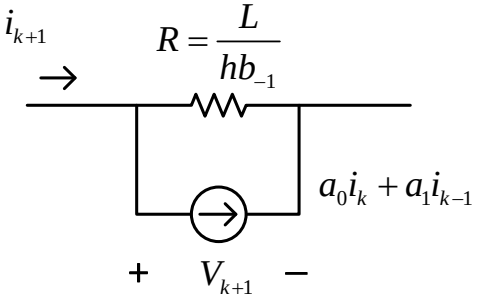
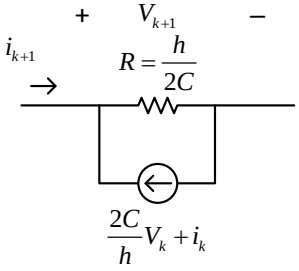
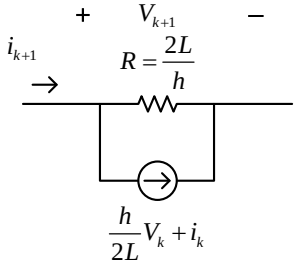
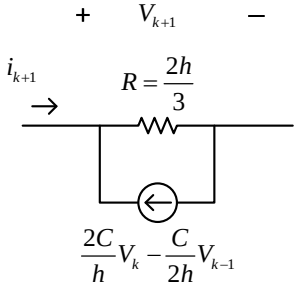
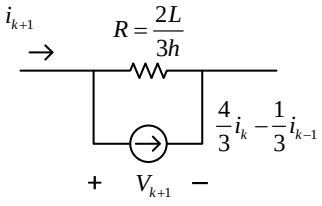


Figure 18 2nd-order Gear companion circuit model for an ideal inductor L

4.3 Companion Model Summary

The previous results are summarized in Table 1. The information storage requirements tend to favor the trapezoidal method in that they are the same for both capacitors and inductors. A bit more organization is required in using the 2nd-order Gear method.

Table 1 Companion Model Summary for L's and C's

		Required Storage			
		V_k	V_{k-1}	i_k	i_{k-1}
Trapezoid Ideal Capacitor		•		•	
Trapezoid Ideal Inductor		•		•	
2nd Gear Ideal Capacitor		•	•		
2nd Gear Ideal Inductor				•	•

5 Arbitrary Circuit Matrix Formulation

The previous material makes it straight forward to finally capture an arbitrary circuit in matrix form to perform time domain simulation. A nodal circuit analysis will be used based node voltages and admittances. The trapezoidal integration formula will be used in the examples which follow. The node voltages and currents are related by the classical formulation

$$yV = i \tag{49}$$

Step #1: Sequentially number the circuit's nodes

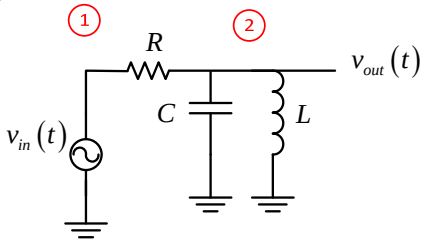


Figure 19 First example problem with nodes numbered sequentially

Step #2: Replace voltage sources by their equivalent Norton current sources

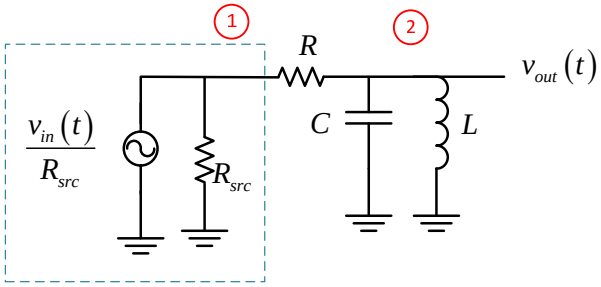


Figure 20 Ideal voltage sources replaced with Norton current-source equivalents (here R_{src} simply made << R)

Step3: Replace all C's and L's by their respective companion model

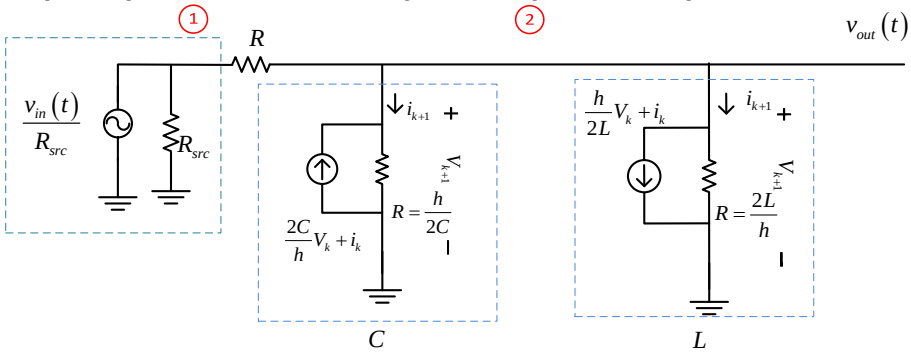


Figure 21 Use of companion models transforms the original circuit into a circuit composed of only resistors and current sources

Step 4: Assemble the admittance matrix y( ) for the circuit

There are only two circuit nodes shown in the example circuit (Figure 21) and 4 resistors. For a given resistor $R = 1/g$ connected between nodes *p* and *q*, the matrix additions are as follows:

$$\begin{aligned} y(p,p) &= y(p,p) + g \\ y(q,q) &= y(q,q) + g \end{aligned} \qquad (50)$$

$$\begin{aligned} y(p,q) &= y(p,q) - g \\ y(q,p) &= y(q,p) - g \end{aligned}$$

For passive circuit elements (R, L, C), this pattern is always the same and is easily repeated for each circuit element. In cases where *p* or *q* is zero (corresponding to ground), the only one of the four equations above will be affected corresponding to the non-zero node index.

The resultant admittance matrix associated with Figure 21 is given by

$$y = \begin{bmatrix} \left(\frac{1}{R_{src}} + \frac{1}{R}\right) & \left(-\frac{1}{R}\right) \\ \left(-\frac{1}{R}\right) & \left(\frac{1}{R} + \frac{2C}{h} + \frac{h}{2L}\right) \end{bmatrix} \qquad (51)$$

Note that this matrix will be unchanged for every time-step iteration! The inverse of the y-matrix will also be constant for every time-step as well.

Step 5: Formulate the current source vector associated with the nodes

Any external current flowing into a circuit node is considered to be negative whereas currents flowing out are positive quantities. It is crucial that this convention be strictly adhered to.

There are 3 current sources shown in Figure 21. Only nodal currents associated with non-ground nodes must be kept track of. The current source vector is given by

$$i_{src_{k+1}} = \begin{bmatrix} \frac{v_{in}}{R_{src}} \\ \left(\frac{2C}{h} v_{out_k} + i_{C_k}\right) - \left(\frac{h}{2L} v_{out_k} + i_{L_k}\right) \end{bmatrix} \qquad (52)$$

where the *C* and *L* subscripts denote capacitor and inductor currents respectively. Aside from computing the inverse of the y-matrix in (51), the iterative computational process can now commence as $V_{k+1} = y^{-1} i_{src_k}$.

The MATLAB script used to iteratively compute the circuit's response is given in Figure 24 with the results shown in Figure 22 and Figure 23 for two different time-steps.

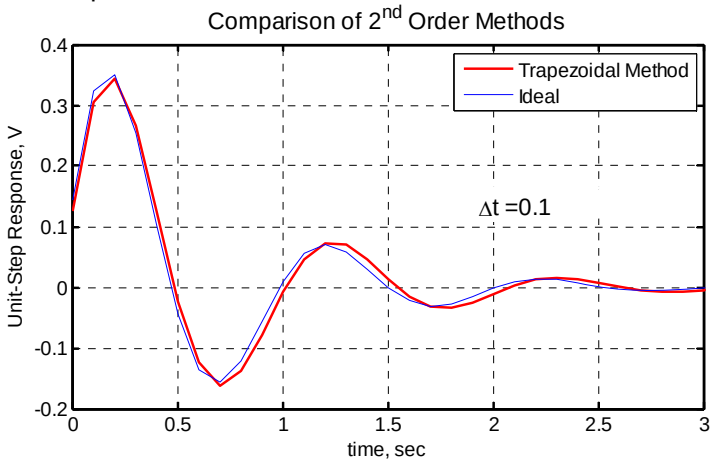


Figure 22 Ideal analytical solution (4) compared with matrix-based solution using Figure 21 with $\Delta t = 0.10$ sec

Figure 23 Ideal analytical solution (4) compared with matrix-based solution using Figure 21 with $\Delta t = 0.01$ sec

5.1 A More Complicated Example

While the simple example just considered is insightful, a more complicated circuit is needed in order to illustrate the MATLAB script formulation more clearly. To this end, consider the complete PLL circuit shown in Figure 25.


```
Rsrc= 0.01;
R= 10;
L= 0.80;
C= 1.0/( (1*2*pi)^2 *L );
dt = 0.01;
ymat= [ (1/Rsrc + 1/R) (-1/R); (-1/R) (1/R + (2*C)/dt + dt/(2*L)) ];
yinv= inv(ymat);
vk= 0.0;
ick= 0.0;
ilk= 0.0;

npts= 5100;
vin= ones(1,npts);
vout= zeros(1,npts);
for ii=1:npts
    isrc= [ vin(ii)/Rsrc, (2*C*vk/dt+ick)-(dt*vk/(2*L)+ilk)]';
    vnodes= yinv*isrc;

    ick= -(2*C*vk/dt + ick);
    ilk= dt*vk/(2*L) + ilk;

    vk= vnodes(2);

    ick= ick + vk*2*C/dt;
    ilk= ilk + vk*dt/(2*L);

    vout(ii)= vk;
end
tm= (0:npts-1)*dt;
p1= plot( tm, vout, 'r' );
set( p1, 'LineWidth', 2 );
```

Figure 24 MATLAB script used to compute the responses in Figure 22 and Figure 23 using the trapezoidal-based companion models

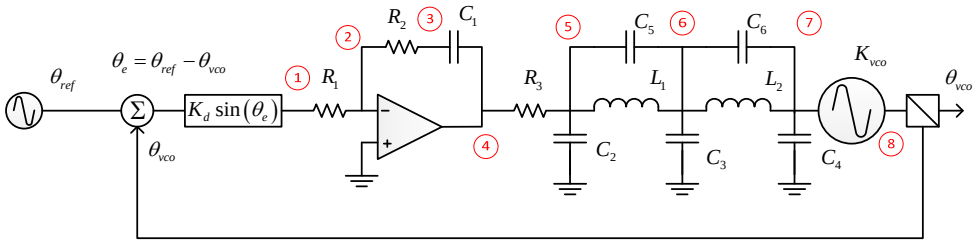


Figure 25 Complete PLL circuit including additional LC elliptic filter and nonlinear sinusoidal phase detector

In this figure, additional steps must be taken to transform it into an equivalent circuit that is easily describable using an admittance matrix. These steps are illustrated in Figure 26 where

1. Voltage output of the op-amp has been transformed into a Norton-equivalent current source;
2. The VCO is recognized as an ideal integrator of phase which has been replaced by a scaled ideal integrator of current which is mathematically equivalent;
3. The output of the phase detector has also been converted to an equivalent Norton source (but not explicitly shown in the schematic)

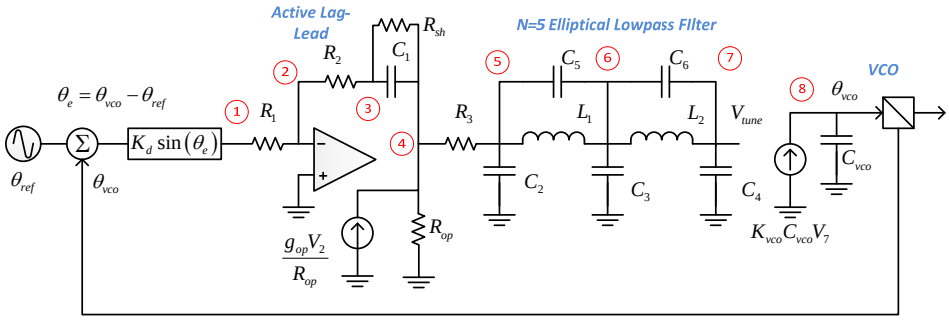


Figure 26 Modified PLL schematic incorporating Norton equivalent circuit for the op-amp output and an additional capacitor C_{vco} as an ideal integrator of phase

Note that the op-amp in Figure 26 has been replaced by an equivalent transconductance amplifier having an output impedance of R_{op}. The transconductance stage is the only non-passive element in the schematic, and compared to the MATLAB admittance formulation described by (50), results in a non-symmetric matrix entry where $y(4,2) = \frac{g_{op}}{R_{op}}$. The MATLAB script for this schematic is provided in the Appendix. A few example transient responses are shown in Figure 27 through Figure 29.

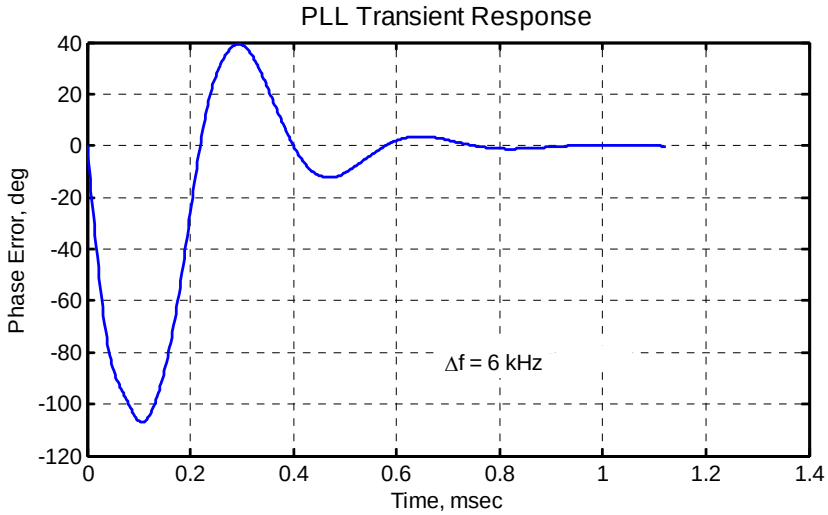


Figure 27 Phase error transient response of the PLL for an initial frequency-step of 6 kHz. Loop natural frequency = 5 kHz with damping factor of 0.707.

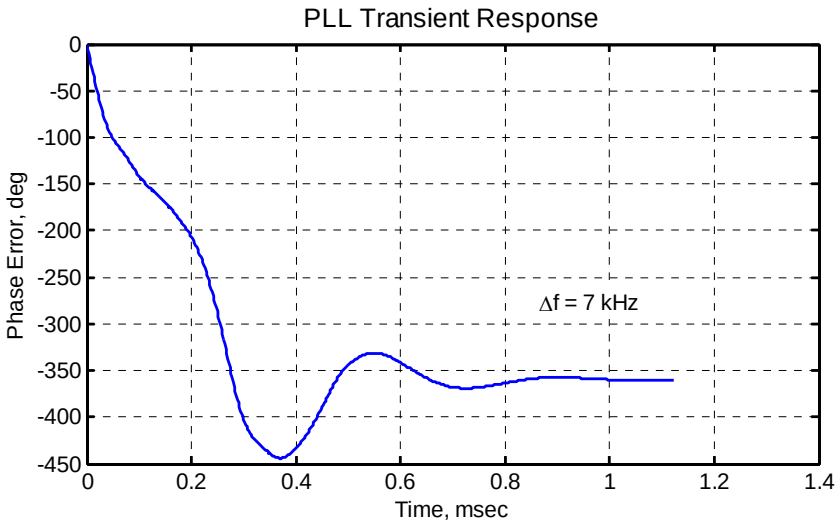


Figure 28 Increasing the initial frequency error to 7 kHz results in a cycle-slip (360°) as shown compared to Figure 27

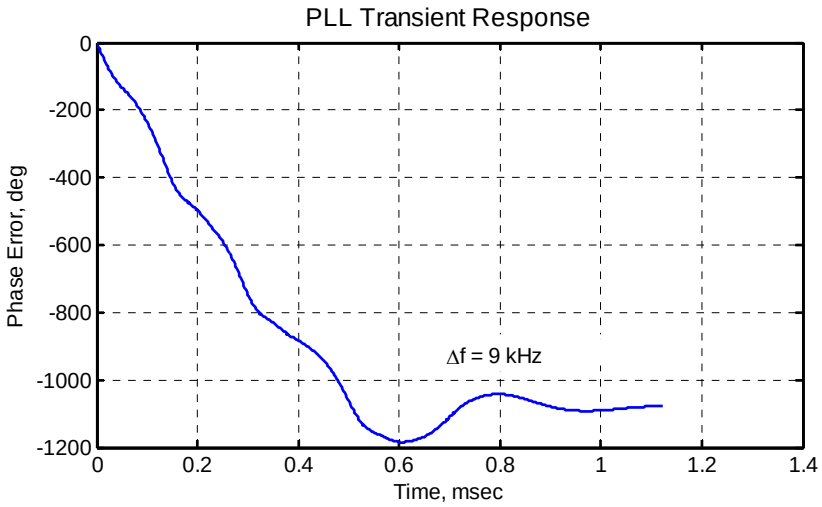


Figure 29 Increasing the initial frequency error to 9 kHz results in more cycle-slipping (1080°) as shown compared to Figure 27

5.2 In Summary

Modeling arbitrary analog and or mixed-signal circuitry in the time domain using MATLAB (or other similar tools like C++) is relatively straight forward using companion models and an admittance matrix. The trapezoidal or 2nd-Order Gear integration formulas are preferred for their precision versus complexity.

Creation of the admittance matrix follows a very structured formula as described earlier and can be done by visual inspection with a little practice. The current source vector assembly requires a bit more care perhaps, but is equally straight forward with some experience. This technique provides an accurate means to capture continuous-time circuit behavior when needed for embedded systems-related analysis.

6 References

1. J.A. Crawford, *Frequency Synthesizer Design Handbook*, Artech House, 1994.
2. Leon O. Chua and Pen-Min Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*, Prentice-Hall, 1975.

7 Appendix:

```
%==================== u23915_timedomain_pll.m ============================
%
%   Example time-domain simulation case based upon
%   trapezoidal companion circuit models
%
disp( ' ' );
disp( ' ' );
Kd= 1;                  % Phase detector gain, V/rad
Kvco= 2*pi*10000;       % VCO gain, rad/sec/V
fn= 5e3;                % Desired natural frequency, Hz
wn= 2*pi*fn;
zeta= 0.707;            % Desired damping factor
%
%   Loop time constant
%
loop_tau= 1/(wn*zeta);
%
%   Do simulation for 25 time constants
%
tsim= 25*loop_tau;
%
%
%   Simulation time increment
%
dt= 0.01/fn;
disp( ['Sampling rate = ' num2str(1/dt)] );
npts= floor( tsim/dt );
tm= (0:npts-1)*dt;      % Time axis

tau1= Kd*Kvco/wn^2;
tau2= 2*zeta/wn;
%
% Resistance level for R1 is arbitrary in this example
%
R1=1e3;
C1= tau1/R1;
R2= tau2/C1;
disp( ['C1 (pF)= ' num2str(C1*1e12) ' R1 = ' num2str(R1)] );
disp( ['R2 = ' num2str(R2) ] );
%
%   Situate 5th order elliptic filter at 100 kHz
%   Pick 20 degree 5% reflection coefficient filter
%
%   Normalized RLC quantities are:
%
R3= 1.0;
C2= 0.3419;
C3= 1.2080;
C4= 1.1545;
C5= 0.0462;
C6= 0.0833;
L1= 0.9108;
L2= 1.2782;
%
%   Scale the normalized elliptic filter RLC's to
%   1 kOhms and a ripple bandwidth of 25 kHz
%
Rscale= 1000;
Fscale= 2*pi*25e3;
%
R3= R3*Rscale;
C2= C2/Rscale/Fscale;
C3= C3/Rscale/Fscale;
```



```
C4= C4/Rscale/Fscale;
C5= C5/Rscale/Fscale;
C6= C6/Rscale/Fscale;
L1= L1*Rscale/Fscale;
L2= L2*Rscale/Fscale;
%
%   Arbitrarily choose Cvco
%   Simply used for scaling within the mathematical model
%
Cvco= 1e-6;
%
%   Choose Op-Amp parameters
%
Rop= 0.01;      % Op-amp's output impedance, Ohms
gop= 1000;      % Assumed transconductance for op-amp
%
%   Choose Norton equivalent output resistance for phase detector
%
Rpd= 0.1;
%
%   Add in shunt resistor across C1
%
Rsh= 1e7;
%- - - - - - - - - - - - - - - - - - - - - - - - - -
%
%   Set up admittance matrix
%   Does not change during iterations
%
%- - - - - - - - - - - - - - - - - - - - - - - - - -
ymat= zeros(8,8);
%
%   Output impedance of phase detector
%
ymat(1,1)= ymat(1,1) + 1/Rpd;
%
%   Entries for R1
%
ymat(1,1)= ymat(1,1) + 1/R1;
ymat(2,2)= ymat(2,2) + 1/R1;
ymat(1,2)= ymat(1,2) - 1/R1;
ymat(2,1)= ymat(2,1) - 1/R1;
%
%   Entries for R2
%
ymat(2,2)= ymat(2,2) + 1/R2;
ymat(3,3)= ymat(3,3) + 1/R2;
ymat(2,3)= ymat(2,3) - 1/R2;
ymat(3,2)= ymat(3,2) - 1/R2;
%
%   Entries for R3
%
ymat(4,4)= ymat(4,4) + 1/R3;
ymat(5,5)= ymat(5,5) + 1/R3;
ymat(4,5)= ymat(4,5) - 1/R3;
ymat(5,4)= ymat(5,4) - 1/R3;
%
%   Output impedance of op-amp
%
ymat(4,4)= ymat(4,4) + 1/Rop;
%
% Rsh
%
ymat(3,3)= ymat(3,3) + 1/Rsh;
ymat(4,4)= ymat(4,4) + 1/Rsh;
ymat(3,4)= ymat(3,4) - 1/Rsh;
ymat(4,3)= ymat(4,3) - 1/Rsh;
```

```
%
%   Capacitor C1 trapezoid companion model
%
ymat(3,3)= ymat(3,3) + 2*C1/dt;
ymat(4,4)= ymat(4,4) + 2*C1/dt;
ymat(3,4)= ymat(3,4) - 2*C1/dt;
ymat(4,3)= ymat(4,3) - 2*C1/dt;
%
%   Capacitor C2
%
ymat(5,5)= ymat(5,5) + 2*C2/dt;
%
%   Capacitor C3
%
ymat(6,6)= ymat(6,6) + 2*C3/dt;
%
%   Capacitor C4
%
ymat(7,7)= ymat(7,7) + 2*C4/dt;
%
%   Capacitor C5
%
ymat(5,5)= ymat(5,5) + 2*C5/dt;
ymat(6,6)= ymat(6,6) + 2*C5/dt;
ymat(5,6)= ymat(5,6) - 2*C5/dt;
ymat(6,5)= ymat(6,5) - 2*C5/dt;
%
%   Capacitor C6
%
ymat(6,6)= ymat(6,6) + 2*C6/dt;
ymat(7,7)= ymat(7,7) + 2*C6/dt;
ymat(6,7)= ymat(6,7) - 2*C6/dt;
ymat(7,6)= ymat(7,6) - 2*C6/dt;
%
%   Cvco
%
ymat(8,8)= ymat(8,8) + 2*Cvco/dt;
%
%   Inductor L1
%
ymat(5,5)= ymat(5,5) + dt/(2*L1);
ymat(6,6)= ymat(6,6) + dt/(2*L1);
ymat(5,6)= ymat(5,6) - dt/(2*L1);
ymat(6,5)= ymat(6,5) - dt/(2*L1);
%
%   Inductor L2
%
ymat(6,6)= ymat(6,6) + dt/(2*L2);
ymat(7,7)= ymat(7,7) + dt/(2*L2);
ymat(6,7)= ymat(6,7) - dt/(2*L2);
ymat(7,6)= ymat(7,6) - dt/(2*L2);
%
%   Transconductance of op-amp
%
ymat(4,2)= gop/Rop;
%
%   VCO tuning
%
ymat(8,7)= -Kvco*Cvco;
%
%   Admittance matrix inverse
%
disp( ['ymat determinant = ' num2str( det(ymat) ) ] );
yinv= inv(ymat);
%
Vk= zeros(1,8);       % Node voltages
ick= zeros(1,7)';     % Capacitor currents
```

```
ilk= zeros(1,2)';      % Inductor currents
ik= zeros(1,8)';
theta_ref= zeros(1,npts)';

%
%   Assume step frequency error at PLL input
%
ferror= 9e3;
theta_ref(1:npts)= 2*pi*ferror*dt*(0:npts-1);  % Phase perturbation to be tracked

theta_vco(1:npts)= 0;
xpts= zeros(1,npts);        % Plotting
%
%    Main iteration loop
%
disp( ['dt = ' num2str(dt)] );
for ii=2:npts
   theta_e= theta_vco(ii-1) - theta_ref(ii);
   Vpd= Kd*sin( theta_e );
   %
   %   Compute capacitor currents
   %
   ick(1)= ick(1) + 2*C1*( Vk(3)-Vk(4) )/dt;
   ick(2)= ick(2) + 2*C2*Vk(5)/dt;
   ick(3)= ick(3) + 2*C3*Vk(6)/dt;
   ick(4)= ick(4) + 2*C4*Vk(7)/dt;
   ick(5)= ick(5) + 2*C5*( Vk(5)-Vk(6) )/dt;
   ick(6)= ick(6) + 2*C6*( Vk(6)-Vk(7) )/dt;
   ick(7)= ick(7) + 2*Cvco*Vk(8)/dt;
   %
   %   Inductor currents
   %
   ilk(1)= ilk(1) + dt*( Vk(5)-Vk(6) )/(2*L1);
   ilk(2)= ilk(2) + dt*( Vk(6)-Vk(7) )/(2*L2);
   %
   %   Set up all node current values associated with companion models
   %
   ik= zeros(1,8)';
   %
   %
   %   Node 1
   %
   ik(1)= Vpd/Rpd;
   %
   %   Node 2
   %
   ik(2)= 0;
   %
   %   Node 3
   %
   ik(3)= ick(1);
   %
   %   Node 4
   %
   ik(4)= -ick(1);
   %
   %   Node 5
   %
   ik(5)= ick(5);
   ik(5)= ik(5) - ilk(1);
   ik(5)= ik(5) + ick(5);
   %
   %   Node 6
   %
   ik(6)= ick(3);
   ik(6)= ik(6) + ilk(1);
```

```
    ik(6)= ik(6) - ick(5);
    ik(6)= ik(6) + ick(6);
    ik(6)= ik(6) - ilk(2);
    %
    %   Node 7
    %
    ik(7)= ick(4);
    ik(7)= ik(7) + ilk(2);
    ik(7)= ik(7) - ick(6);
    %
    %   Node 8
    %
    ik(8)= ick(7);
    %
    %   Compute new node voltages
    %
    theta_vco(ii)= Vk(8);
    Vk= yinv*ik;
    xpts(ii)= theta_e;
    %
    %   Compute new capacitor current values
    %
    ick(1)= -ick(1) + 2*C1*( Vk(3)-Vk(4) )/dt;
    ick(2)= -ick(2) + 2*C2*Vk(5)/dt;
    ick(3)= -ick(3) + 2*C3*Vk(6)/dt;
    ick(4)= -ick(4) + 2*C4*Vk(7)/dt;
    ick(5)= -ick(5) + 2*C5*( Vk(5)-Vk(6) )/dt;
    ick(6)= -ick(6) + 2*C6*( Vk(6)-Vk(7) )/dt;
    ick(7)= -ick(7) + 2*Cvco*Vk(8)/dt;
    %
    %   Compute new inductor currents
    %
    ilk(1)= ilk(1) + dt*( Vk(5)-Vk(6) )/(2*L1);
    ilk(2)= ilk(2) + dt*( Vk(6)-Vk(7) )/(2*L2);
end

fig1= figure(1);
clf;
axes( 'FontName', 'Arial', 'FontSize', 12 );
p1= plot( tm/0.001, xpts*180/pi, 'b' );
set( p1, 'LineWidth', 2 );
xlabel( 'Time, msec', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Phase Error, deg', 'FontName', 'Arial', 'FontSize', 12 );
title( 'PLL Transient Response', 'FontName', 'Arial', 'FontSize', 14 );
h= gca;
set( h, 'LineWidth', 2 );
grid on
txt= strcat( ['\Deltaf = ', num2str(ferror*0.001), ' kHz'] );
annotation(fig1,'textbox','String',{txt},'FontSize',12,...
    'FontName','Arial',...
    'FitBoxToText','off',...
    'LineStyle','none',...
    'BackgroundColor',[1 1 1],...
    'Position',[0.504 0.2769 0.2112 0.06183]);
```

[i] Version 1.0, October 10, 2016
[ii] From u23908_stability_regions.m.
[iii] From u23908_stability_regions.m.
[iv] Ibid.