Using Mathcad To Derive Circuit Equations and Optimize Circuit Behavior

By:
James C. (Jim) Bach
EE Analysis Engineer & Mathcad Instructor
Electrical Design & Analysis Group
Delphi Corporation
Kokomo, IN, USA
James.C.Bach@Delphi.com

Introduction

For those of you who are not familiar with it, Mathcad (produced by Mathsoft, not to be confused with Matlab or Mathematica) is a general purpose mathematical analysis tool with WYSIWYG formula entry and which can operate in both numeric and symbolic modes; symbolic math is performed by the underlying Maple engine, common to many of these "Math Processor" applications. Equations are entered in a natural, 'text book' format wherein integrals look like integrals, and summations look like summations; FORTRAN-like coding syntaxes need not apply. It is a Swiss-army knife of mathematical functions, operators, GUI widgets, and graph types. It comprehends (and automatically converts between) units, meaning that when you divide a "Volts" quantity by an "Amps" quantity, your result is automatically an "Ohms" quantity. From the EE's point of view, Mathcad is a tool that can be used to derive system- or circuit-level equations (remember "solving N equations with N unknowns"?), optimize circuit component values, perform worst-case analysis, process large vectors (matrices) of data, perform image-processing and signal-processing tasks, curve-fit lab-collected measurements, and even create .AVI movies (animations of data). After some exploration and experimentation with Mathcad I can guarantee you that you will start pushing-aside faithful old Excel. Speaking of which, Mathcad allows you to embed Excel tables inside its documents, so that users can easily enter data into variables, or display calculation results in tabular form; Mathcad can even read/write data from/to Excel files in the native binary .XLS format. What could make the transition from one tool to the other easier?

This article provides a brief tutorial on Mathcad's "Symbolics" and "Optimization" features, and then illustrates how to use both of them to perform a circuit design. For those of you who might already use Mathcad some of this might be "review"; perhaps, though, you will pick-up some new tidbits of knowledge. It is hoped that the reader will learn a faster and easier (and less error-prone) method of deriving circuit equations and transfer functions, as well as how to automatically determine optimal component values based on end-product requirements. It is also hoped that this article will expose the reader to a new analytical tool (Mathcad) that he wasn't already aware of, and possibly open the door to many new opportunities for performing better design and analysis tasks in a faster and more efficient manner.

Symbolic Math In Mathcad

For the EE design engineer, one of Mathcad's strong points is its "Symbolic Math" processor. In its simplest form this feature allows you to convert or mutate an equation into another form. For instance, say we take our familiar "impedance of 2 paralleled resistors" equation and wish to rearrange the equation so we can solve for one of the resistor values ($\mathbf{R_1}$) if we know the combined resistance ($\mathbf{Z_{Parallel}}$) and one of the two resistor values ($\mathbf{R_2}$). Using the two simplest forms of symbolic processing in Mathcad, which I call "Static Symbolics" and "Live Symbolics":

Static: Using the "**Symbolics→Variable→Solve**" menu operation, with $\mathbf{R_1}$ selected:

$$Z_{Parallel} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}} \quad \text{has solution(s)} \quad Z_{Parallel} \cdot \frac{R_2}{-Z_{Parallel} + R_2}$$

Live: Using the "**Solve**" operator from the "Symbolics Toolbar":

$$Z_{Parallel} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}} \text{ solve}, R_1 \rightarrow -Z_{Parallel} \cdot \frac{R_2}{Z_{Parallel} - R_2}$$

Or how about rearranging the familiar "resonant frequency of an LC 'Tank' circuit"? Again, using the simplest forms of symbolic processing in Mathcad we can easily rearrange the equation to calculate the capacitance ($\mathbf{C_{Cap}}$) needed to make a given inductor ($\mathbf{L_{Coil}}$) resonate at a certain frequency ($\mathbf{F_{Res}}$):

Static: Using the "**Symbolics→Variable→Solve**" menu operation, with $\mathbf{C_{Cap}}$ selected:

$$F_{Res} = \frac{1}{2 \cdot \pi \cdot \sqrt{L_{Coil} \cdot C_{Cap}}} \quad \text{has solution(s)} \quad \frac{1}{4 \cdot L_{Coil} \cdot F_{Res}^2 \cdot \pi^2}$$

Live: Using the "**Solve**" operator from the "Symbolics Toolbar":

$$F_{Res} = \frac{1}{2 \cdot \pi \cdot \sqrt{L_{Coil} \cdot C_{Cap}}} \text{ solve}, C_{Cap} \rightarrow \frac{1}{4 \cdot L_{Coil} \cdot F_{Res}^2 \cdot \pi^2}$$

How does one utilize these two simple forms of symbolics? Well, the "Static" symbolics method uses the various operators located under the top-line pull-down menu titled "Symbolics". In general, you type-in an equation, select a term (variable) in the equation, and then select the symbolic operation you wish to perform; some operations will operate on the entire equation regardless of what is selected. For instance, creating the $\mathbf{R_1}$ derivation takes-place as:

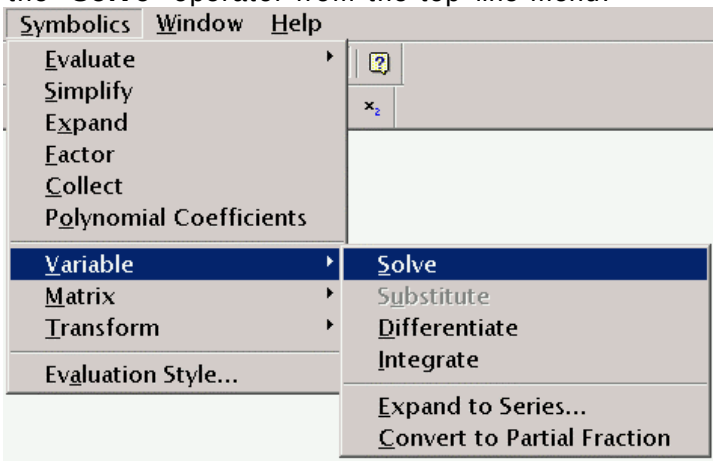
Enter the equation:

$$Z_{Parallel} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

Select the R_1 term (with 'blue L-shaped underline'):

$$\frac{1}{R_1}$$

Invoke the "**Solve**" operator from the top-line menu:

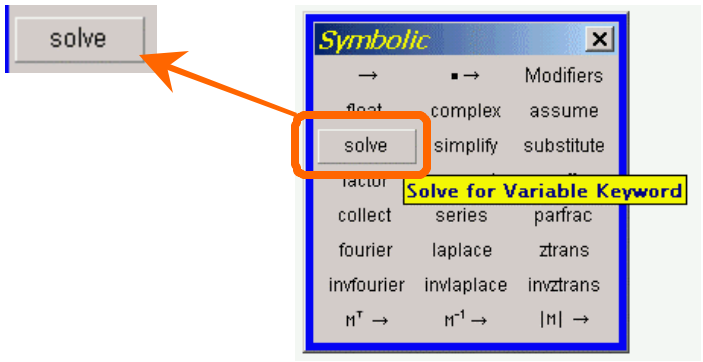


The result is inserted into the document:

has solution(s) $Z_{Parallel} \cdot \frac{R_2}{-Z_{Parallel} + R_2}$

The "Live" symbolics method uses the various operators on the "Symbolic" toolbar. In general, you place your insertion mark where (in your document) you want your symbolic evaluation to take place, select the symbolic operator from the toolbar, and then fill-in the placeholders (little black squares) with your equation and the term to be solved for. For instance, creating the $\mathbf{C_{cap}}$ derivation takes-place as:

Click the "Solve" icon on the toolbar:



The "**Solve**" operation is dropped-in, with placeholders ready and waiting:

$\blacksquare\ \text{solve}, \blacksquare \rightarrow$

User fills-in the left-most placeholder (small square) with the original equation:

$F_{Res} = \frac{1}{2 \cdot \pi \cdot \sqrt{L_{Coil} \cdot C_{Cap}}}\ \text{solve}, \blacksquare \rightarrow$

User fills-in the right-most placeholder with the variable (term) to be solved-for:

$F_{Res} = \frac{1}{2 \cdot \pi \cdot \sqrt{L_{Coil} \cdot C_{Cap}}}\ \text{solve}, C_{Cap} \rightarrow$

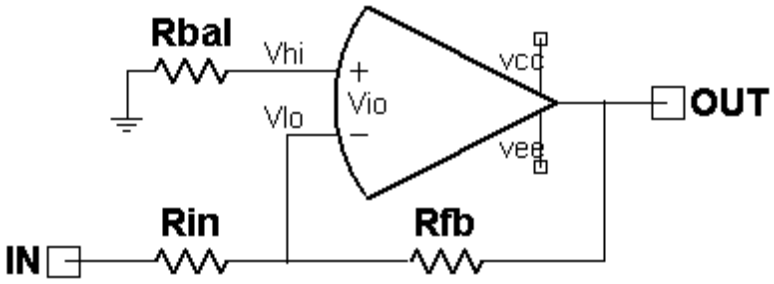
Using Mathcad To Derive Circuit Equations and Optimize Circuit Behavior

Mathcad automatically calculates the answer (right-side of → symbol):

$$F_{Res} = \frac{1}{2\cdot\pi\cdot\sqrt{L_{Coil}\cdot C_{Cap}}} \text{ solve}, C_{Cap} \rightarrow \frac{1}{4\cdot L_{Coil}\cdot F_{Res}^{2}\cdot\pi^{2}}$$

The more powerful "Chained Live" symbolic processor allows you to combine a set of equations, solving for multiple variables contained in the equations; remember the not-so-fun task of "solving N equations with N unknowns"? In addition, that symbolic result can be assigned to a numeric function that can be used in subsequent design/analysis equations. For example, you can write nodal equations for a circuit and have Mathcad derive symbolic (algebraic) equations that represent the voltages at the nodes (or currents through the elements), and then assign the equation representing the **OUT** node to a function that can be used to plot the transfer function. A quick example is shown below.

Given a simple inverting amplifier using an op-amp:



Derive the nodal equations, presuming input currents are zero (ideal op-amp), but provide a term for the input offset voltage (**Vio**):

Eqn1 := $\mathbf{Vio} = \mathbf{Vhi} - \mathbf{Vlo}$ *Input offset voltage (DV across + and - pins)*

Eqn2a := $\mathbf{I_{Hi}} = 0$ **Eqn2b** := $\mathbf{I_{Lo}} = 0$ *Input currents (are zero for idealistic case)*

Eqn3a := $\mathbf{I_{Hi}} = \frac{\mathbf{Vhi}}{\mathbf{Rbal}}$ *Current through Rbal (from "+" pin)*

Eqn3b := $\mathbf{I_{Lo}} = \frac{\mathbf{Vlo} - \mathbf{IN}}{\mathbf{Rin}} + \frac{\mathbf{Vlo} - \mathbf{OUT}}{\mathbf{Rfb}}$ *Current at Vlo node (from "-' pin)*

Use the symbolics processor to calculate the equation for the output voltage (and a bunch of intermediary terms that we don't care about):

$$\mathbf{SysRes} := \begin{pmatrix} \mathbf{Eqn1} \\ \mathbf{Eqn2a} \\ \mathbf{Eqn2b} \\ \mathbf{Eqn3a} \\ \mathbf{Eqn3b} \end{pmatrix} \text{solve}, \begin{pmatrix} \mathbf{OUT} \\ \mathbf{Vlo} \\ \mathbf{Vhi} \\ \mathbf{I_{Lo}} \\ \mathbf{I_{Hi}} \end{pmatrix} \rightarrow \begin{bmatrix} \frac{-(\mathbf{Rfb}\cdot\mathbf{Vio} + \mathbf{Rfb}\cdot\mathbf{IN} + \mathbf{Rin}\cdot\mathbf{Vio})}{\mathbf{Rin}} & -\mathbf{Vio} & 0 & 0 & 0 \end{bmatrix}$$

Strip-out the one answer we care about (**OUT**), and collect on the **Vio** term to make it 'neater looking':

$$\mathbf{Out} := \left(\mathbf{SysRes}^{\langle 0 \rangle}\right)_0 \text{ collect}, \mathbf{Vio} \rightarrow \frac{-(\mathbf{Rfb} + \mathbf{Rin})}{\mathbf{Rin}}\cdot\mathbf{Vio} - \mathbf{Rfb}\cdot\frac{\mathbf{IN}}{\mathbf{Rin}}$$

Then assign the symbolic result to a function (V_{Out}) that can be evaluated numerically:

$$V_{Out}(IN, Rfb, Rin, Vio) := Out \rightarrow \frac{-(Rfb + Rin)}{Rin} \cdot Vio - Rfb \cdot \frac{IN}{Rin}$$

The above is an example of what I call "Chained Live Symbolics", because the equations are tied-together or "chained" by virtue of assigning the symbolic equations (right-side of := operator) to variables (left-side of := operator), which are then used in subsequent symbolic operations. For instance, the 5 basic circuit equations, assigned into variables **Eqn1**, **Eqn2a**, **Eqn2b**, **Eqn3a**, and **Eqn3b**, are chained to (used by) the 'solve' operation, whose results are assigned into the variable **SysRes** (System Results), which is then chained to (used by) the equation that pulls-out the OUT answer (and collects terms), and assigns it to the variable **Out**, which is chained to (used by) the function declaration for V_{Out}. Notice that our final result does not contain a term for **Rbal**; because we had defined our op-amp's input currents to be zero, the voltage (**Vhi**) induced into **Rbal** from the "+" input's current is zero, thus the **Rbal** term drops-out of our transfer function.

The 'beauty' of this system is that if you find an error in one of your fundamental equations, or wish to change one or more equations because you made a topological change in the circuit, all of the changes automatically ripple-down to the bottom-line answer. For instance, the example above ignored the op-amp's input offset and bias currents. If we later decided to add terms (**Io** and **Ib**, respectively) to account for these parasitics, all we would need to do is modify equations **Eqn2a** and **Eqn2b** as follows:

$$Eqn2a := I_{Hi} = Ib + \frac{1}{2} \cdot Io \qquad Eqn2b := I_{Lo} = Ib - \frac{1}{2} \cdot Io$$

And then the new System Results pops-out automatically (and in a much more 'long and messy' form):

$$SysRes = \begin{pmatrix} Eqn1 \\ Eqn2a \\ Eqn2b \\ Eqn3a \\ Eqn3b \end{pmatrix} solve, \begin{pmatrix} OUT \\ Vio \\ Vhi \\ I_{Lo} \\ I_{Hi} \end{pmatrix} \rightarrow \left(\frac{-1}{2} \cdot \frac{2 \cdot Rin \cdot Rfb \cdot Ib - Rin \cdot Rfb \cdot Io + 2 \cdot Rfb \cdot Vio - 2 \cdot Rfb \cdot Rbal \cdot Ib - Rfb \cdot Rbal \cdot Io + 2 \cdot Rfb \cdot IN + 2 \cdot Rin \cdot Vio - 2 \cdot Rin \cdot Rbal \cdot Ib - Rin \cdot Rbal \cdot Io}{Rin} \right. \left. - Io + Rbal \cdot Ib + \frac{1}{2} \cdot Rbal \cdot Io \quad Rbal \cdot Ib + \frac{1}{2} \cdot Rbal \cdot Io \quad Ib - \frac{1}{2} \cdot Io \quad Ib + \frac{1}{2} \cdot Io \right)$$

$$, \begin{pmatrix} OUT \\ Vio \\ Vhi \\ I_{Lo} \\ I_{Hi} \end{pmatrix} \rightarrow \left(\frac{-1}{2} \cdot \frac{2 \cdot Rin \cdot Rfb \cdot Ib - Rin \cdot Rfb \cdot Io + 2 \cdot Rfb \cdot Vio - 2 \cdot Rfb \cdot Rbal \cdot Ib - Rfb \cdot Rbal \cdot Io + 2 \cdot Rfb \cdot IN + 2 \cdot Rin \cdot Vio - 2 \cdot Rin \cdot Rbal \cdot Ib - Rin \cdot Rbal \cdot Io}{Rin} \right.$$

After collecting on terms **Vio**, **Ib**, and **Io** the V_{Out} function declaration changes to:

$$V_{Out}(IN, Rfb, Rin, Rbal, Vio, Ib, Io) \doteq Out \rightarrow \frac{1}{2} \cdot \frac{-2 \cdot Rfb - 2 \cdot Rin}{Rin} \cdot Vio + \frac{1}{2} \cdot \frac{-2 \cdot Rin \cdot Rfb + 2 \cdot Rin \cdot Rbal + 2 \cdot Rfb \cdot Rbal}{Rin} \cdot Ib + \frac{1}{2} \cdot \frac{Rin \cdot Rfb + Rin \cdot Rbal + Rfb \cdot Rbal}{Rin} \cdot Io - Rfb \cdot \frac{IN}{Rin}$$

All we had to do is tidy-up V_{Out}'s argument list by adding-in the two new terms, **Io** and **Ib**. It was optional to collect on the newly-added terms in order to make the equation look prettier, however, notice how it lends itself to allowing the designer to directly observe the contributions of the op-amp's parasitics in the overall transfer function. In fact, you can

even use Mathcad's "derivative" operator to directly find the individual contributions of these error terms to the transfer function:

$$\frac{d}{dVio}Out \rightarrow \frac{-1}{2}\cdot\frac{2\cdot Rfb + 2\cdot Rin}{Rin}$$

$$\frac{d}{dIb}Out \rightarrow \frac{-1}{2}\cdot\frac{2\cdot Rin\cdot Rfb - 2\cdot Rin\cdot Rbal - 2\cdot Rfb\cdot Rbal}{Rin}$$

$$\frac{d}{dIo}Out \rightarrow \frac{-1}{2}\cdot\frac{-Rin\cdot Rfb - Rin\cdot Rbal - Rfb\cdot Rbal}{Rin}$$

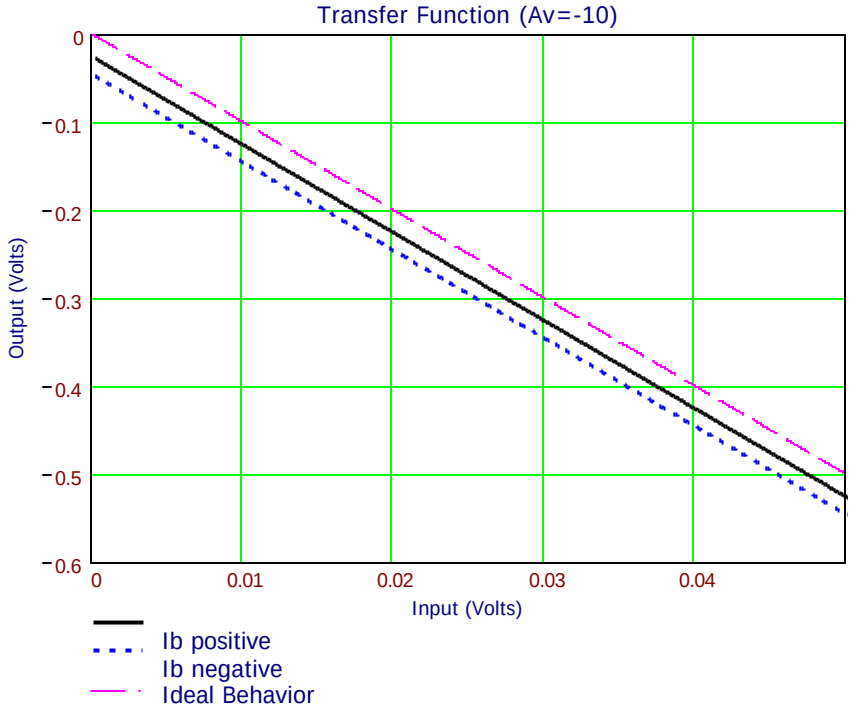
And, of course, the "gain" of the circuit is simply the derivative of the transfer function with respect to the input signal "**IN**":

$$\frac{d}{dIN}Out \rightarrow \frac{-Rfb}{Rin}$$

Notice that our final result (**OUT**) now contains terms involving **Rbal**; because we have included terms (Ib and Io) which generate current ($\mathbf{I_{Hi}}$) on the "+" input, the voltage (**Vhi**) induced across **Rbal** is no longer zero, thus **Rbal** is needed in the transfer function.

So now we have a function that can calculate the output voltage (**OUT**) given all of the circuit values and device parasitics. Using this function we can (amongst other things) plot a graph showing the circuit's transfer function. Say we've designed this circuit to have a gain of –10, and we are using an op-amp with fairly large input currents:

$R_{fb} := 10k\Omega$ $R_{in} := 1k\Omega$ $R_{bal} := 1k\Omega$ $V_{io} := 5mV$ $I_b := 10\mu A$ $I_o := 0.1\mu A$



Optimization in Mathcad

Another one of Mathcad's strong points is that it has built-in "optimization" capability, which can be used to adjust any number of system variables until a set of "goal" conditions have been met, or met with minimal error. This allows the engineer to make some initial guesses for component values, and then have Mathcad determine what the optimal values are in order to meet the design constraints (say, to reduce the error of a current-sense amplifier, obtain a desired frequency response in a multi-stage filter network, etc.). The designer can constrain component values to particular ranges, to prevent the optimizer from finding problematically "too small" or "too large" of values. The designer can also create his own "Error" function, which can be used to control the weighting of trade-offs in the optimization process.

First of all, you must know if the system you are trying to optimize has achievable goals (constraints), or if you have some mutually exclusive goals that will make an exact solution impossible. Mathcad provides two constructs for optimization:

Given…Find	Finds an exact solution (makes all constraints come true)
Given…Minerr	Finds the solution with minimal error for all constraints

If you know your system DOES have an exact solution, then use the **Given…Find** construct. Otherwise use the **Given…Minerr** construct, which will provide a solution whether or not an exact solution exists. The following examples demonstrate how each of these constructs work.

The first example of optimization illustrates use of the **Given…Find** construct to determine the optimal values of two resistors in a divider such that both the target output voltage and target Thevenin resistance are obtained. First we need to create functions for calculating the output voltage (**V_{Out}**) and the Thevenin resistance (**Z_{Out}**), specify the supply voltage (**Vs**), and establish the constraints on the system (**$Goal_V$** and **$Goal_Z$**):

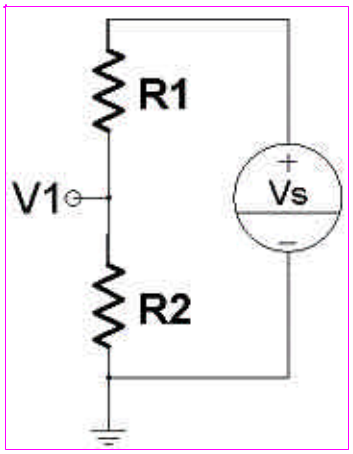
$$V_{Out}(V_S, R_1, R_2) := V_S \cdot \frac{R_2}{R_1 + R_2}$$

Output Voltage

Functions describing circuit behavior to be optimized

$$Z_{Out}(R_1, R_2) := \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

Output Impedance



$V_{Supply} := 5V$ *Power Supply Voltage*

Voltage

Optimization Goals

$Goal_V := 1.5V$

$Goal_Z := 10k\Omega$ *Impedance*

Next we make our "guesses" for the values of R1 and R2:

$R_1 := 2Goal_Z$ $R_2 := R_1$

Then we let Mathcad perform the optimization, telling it to force $\mathbf{V_{Out}}$ to match $\mathbf{Goal_V}$ and to force $\mathbf{Z_{Out}}$ to match $\mathbf{Goal_Z}$:

Given

$V_{Out}(V_{Supply}, R_1, R_2) = Goal_V \qquad Z_{Out}(R_1, R_2) = Goal_Z$

$\begin{pmatrix} R_{1\_Found} \\ R_{2\_Found} \end{pmatrix} := Find(R_1, R_2)$

It is important to note that the "**=**" signs used to define the constraints of the **Given…Find** block are the "Boolean Equals" sign, obtained by clicking the icon on the "Boolean Toolbar" (or typing **<ctrl>=**); this is NOT the standard "give me the answer" equals sign obtained by typing "=" on the keyboard.

The results that Mathcad yields are:

$R_{1\_Found} = 33.3333\,k\Omega$

$R_{2\_Found} = 14.2857\,k\Omega$

Calling our original functions with the newly found resistor values we can check compliance with the initial design constraints:

$V_{Out}(V_{Supply}, R_{1\_Found}, R_{2\_Found}) = 1.5000$

$Z_{Out}(R_{1\_Found}, R_{2\_Found}) = 10.0000\,k\Omega$

Indeed, we see that our goals have been met.

We can generalize this optimization so that no matter how many different $\mathbf{V_{Out}}$ and $\mathbf{Z_{Out}}$ combinations we need to create, we don't have to recreate the **Given…Find** block multiple times. In this example we'll always default the resistors to 10kΩ, and we'll replace the left-side of the **:= Find** equation with the name of a function and a list of input arguments:

$R_1 := 10k\Omega \qquad R_2 := 10k\Omega$

Given

$V_{Out}(V_S, R_1, R_2) = T_V \qquad Z_{Out}(R_1, R_2) = T_Z$

$Optimize\_Rs1(V_S, T_V, T_Z) := Find(R_1, R_2)$

What this allows us to do is pass-in any set of target $\mathbf{V_{Out}}$ and $\mathbf{Z_{Out}}$, and get-back a set of $\mathbf{R_1}$ and $\mathbf{R_2}$. For example:

$Optimize\_Rs1(5V, 3V, 1k\Omega) = \begin{pmatrix} 1666.6655 \\ 2499.9986 \end{pmatrix}\Omega$

$Optimize\_Rs1(12V, 3V, 1k\Omega) = \begin{pmatrix} 4000.0009 \\ 1333.3333 \end{pmatrix}\Omega$

$Optimize\_Rs1(12V, 5V, 10k\Omega) = \begin{pmatrix} 24.0000 \\ 17.1429 \end{pmatrix}k\Omega$

The second example of optimization illustrates use of the **Given…Minerr** construct to determine the optimal values of three resistors in a divider such that both of the target output voltages are obtained. The list of constraints includes an inane requirement for the top and bottom resistor values to be the same value; this precludes the **Given…Find** construct from succeeding.

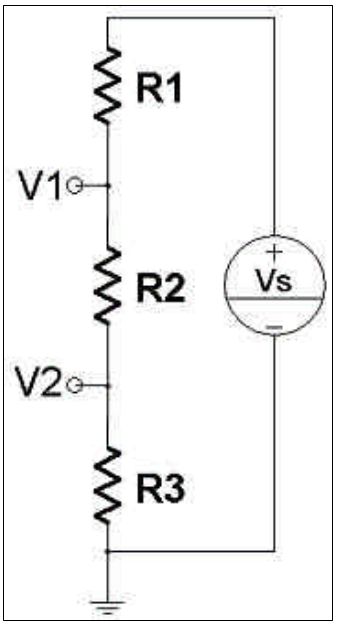
Using Mathcad To Derive Circuit Equations and Optimize Circuit Behavior

First we need to create functions for calculating the output voltages (**V_{Out1}** and **V_{Out2}**), specify the supply voltage (**Vs**), and establish the constraints on the system (**$Goal_{V1}$** and **$Goal_{V2}$**):

$$V_{Out1}(V_S, R_1, R_2, R_3) := V_S \cdot \frac{R_2 + R_3}{R_1 + R_2 + R_3}$$ *Top "Tap"*

Functions describing circuit behavior to be optimized

$$V_{Out2}(V_S, R_1, R_2, R_3) := V_S \cdot \frac{R_3}{R_1 + R_2 + R_3}$$ *Bottom "Tap"*



$V_{Supply} := 5V$ *Power Supply Voltage*

$Goal_{V1} := 4V$ *Top "Tap"*

Optimization Goals

$Goal_{V2} := 1.5V$ *Bottm "Tap"*

Next we make our "guesses" for the values of R1, R2 and R3:

$R_1 := 1k\Omega$ $R_2 := 1k\Omega$ $R_3 := 1k\Omega$

Then we let Mathcad perform the optimization, telling it to force **V_{Out1}** to match **$Goal_{V1}$**, to force **V_{Out2}** to match **$Goal_{V2}$**, and to force **R_1** to match **R_3** (i.e. same-valued resistors):

Given

$V_{Out1}(V_{Supply}, R_1, R_2, R_3) = Goal_{V1}$

$V_{Out2}(V_{Supply}, R_1, R_2, R_3) = Goal_{V2}$

$R_1 = R_3$

$$\begin{pmatrix} R_{1\_Found} \\ R_{2\_Found} \\ R_{3\_Found} \end{pmatrix} := Minerr(R_1, R_2, R_3)$$

Again, please note that the three constraints in the **Given…Minerr** block utilize the "Boolean Equals" sign.

The results that Mathcad yields shows us that it complied with the constraint of $R_1 = R_3$:

$R_{1\_Found} = 0.2500k\Omega$ $R_{2\_Found} = 0.5000k\Omega$ $R_{3\_Found} = 0.2500k\Omega$

However, calling our original functions with the newly found resistor values shows us that the voltage constraints were not quite met:

$V_{Out1}(V_{Supply}, R_{1\_Found}, R_{2\_Found}, R_{3\_Found}) = 3.7500V$ $Goal_{V1} = 4.0000V$

$V_{Out2}(V_{Supply}, R_{1\_Found}, R_{2\_Found}, R_{3\_Found}) = 1.2500V$ $Goal_{V2} = 1.5000V$

Using Mathcad To Derive Circuit Equations and Optimize Circuit Behavior

Because the desired voltages across R_1 and R_3 are different (1V –vs- 1.5V) there is NO way same-valued resistors can be used and provide the desired output voltages. Notice that both of the output voltages missed their targets by the same amount; $\mathbf{V_{Out1}}$ is 0.25V lower than desired and $\mathbf{V_{Out2}}$ is 0.25V higher than desired. The optimizer did its best with conflicting constraints; it "split the difference".

The key to getting even this close to an optimized solution is the use of the "**Given…Minerr**" construct. If we attempted this optimization using the "**Given…Find**" construct, we'd find-out the hard way that we had an impossible situation:

Given

$V_{Out1}(V_{Supply}, R_1, R_2, R_3) = Goal_{V1}$

$V_{Out2}(V_{Supply}, R_1, R_2, R_3) = Goal_{V2}$

$R_1 = R_3$

$$\begin{pmatrix} R_{1\_Found} \\ R_{2\_Found} \\ R_{3\_Found} \end{pmatrix} := Find(R_1, R_2, R_3)$$

No solution was found. Try changing
the guess value or the value of TOL or CTOL.

Whenever you attempt to perform an optimization using the "**Given…Find**" construct, and you get this sort of error, try changing "**Find**" to "**Minerr**" and see what results you get; perhaps there really was NOT an exact solution. In general the "**Given…Minerr**" construct is the sure-bet; it will give an answer whether or not there is an exact solution.

Notice in the example below that if we unshackle the optimizer (remove $\mathbf{R_1=R_3}$ constraint), then the "**Given…Find**" has no problem in finding an exact solution:

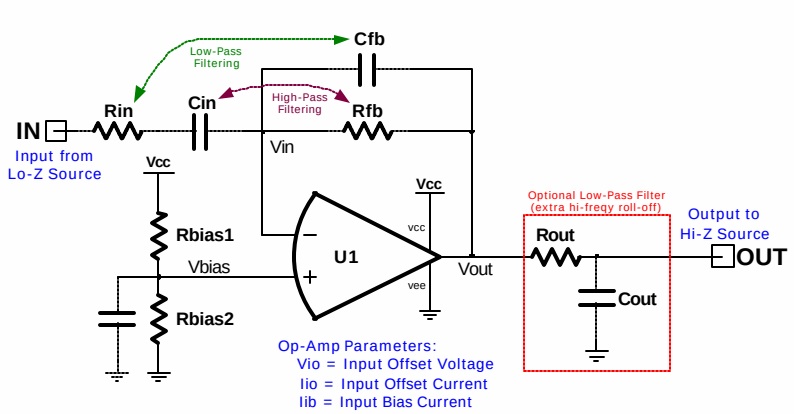
$R_{1\_Found} = 0.5950\,k\Omega$ $R_{2\_Found} = 1.4876\,k\Omega$ $R_{3\_Found} = 0.8926\,k\Omega$

$V_{Out1}(V_{Supply}, R_{1\_Found}, R_{2\_Found}, R_{3\_Found}) = 4.0000\,V$ $Goal_{V1} = 4.0000\,V$

$V_{Out2}(V_{Supply}, R_{1\_Found}, R_{2\_Found}, R_{3\_Found}) = 1.5000\,V$ $Goal_{V2} = 1.5000\,V$

Putting it all together

Now that we know how to use the "Symbolics" processor to derive circuit equations and how to use the "Optimizer" to obtain optimal component values to meet a set of goals (targets), let's put the two techniques together to synthesize a useful design. The example provided here is a real-world circuit, a microphone preamplifier with bandpass filtering. The circuit topology is:



The heart of this circuit topology is op-amp (**U1**), arranged in an inverting amplifier configuration. This analysis shall take into account the input parasitics of the op-amp, namely the input offset voltage (**Vio**), the input offset current (**Iio**), and the input bias current (**Iib**). Because this design uses a single-supply op-amp, the non-inverting ("+") input of the op-amp is held at a pseudo-ground voltage of mid-supply, created by a simple resistor divider network consisting of **Rbias1** and **Rbias2**. Part of the optimization process will be to choose divider resistor values that minimize output offset (i.e. shift from the desired mid-supply value).

The frequency-selectivity of the circuit is controlled by the RC elements in both the input and feedback legs. Input resistor **Rin** and feedback capacitor **Cfb** form a low-pass "Pole", while input capacitor **Cin** and feedback resistor **Rfb** form a high-pass "Zero". Additional low-pass filtering is provided by optional output elements **Rout** and **Cout**, forming another "Pole"; these elements can be eliminated if the required transfer function does not need them.

The first step in our design process is to derive the transfer function for this circuit. To do this we will make use of Mathcad's "Symbolics" processor, just as we did with the simpler inverting amplifier. We begin by writing equations for each of the circuit's reactive elements (capacitors):

$$Eqn_{1a} := X_{Cin} = \frac{1}{2i\pi \cdot Frq \cdot Cin}$$

$$Eqn_{1b} := X_{Cfb} = \frac{1}{2i\pi \cdot Frq \cdot Cfb}$$

$$Eqn_{1c} := X_{Cout} = \frac{1}{2i\pi \cdot Frq \cdot Cout}$$

Then we calculate the combined (complex) impedances of series/parallel RC branches:

$$Eqn_{2a} := X_{IN} = X_{Cin} + Rin$$

$$Eqn_{2b} := X_{FB} = \frac{1}{\frac{1}{X_{Cfb}} + \frac{1}{Rfb}}$$

Then we write equations that describe the op-amp's input characteristics:

$$Eqn_{3a} \doteq Vio = Vbias - Vin$$

$$Eqn_{3b} \doteq I_{Inv} = Iib + \frac{1}{2}Iio$$

$$Eqn_{3c} \doteq I_{NonInv} = Iib - \frac{1}{2}Iio$$

Then we write our nodal equations:

$$Eqn_{4a} \doteq \frac{IN - Vin}{X_{IN}} = \frac{Vin - Vout}{X_{FB}} + I_{Inv}$$

$$Eqn_{4b} \doteq \frac{Vin - Vout}{X_{FB}} = \frac{Vout - OUT}{Rout} + I_{Out}$$

$$Eqn_{4c} \doteq \frac{Vout - OUT}{Rout} = \frac{OUT}{X_{Cout}}$$

$$Eqn_{4d} \doteq \frac{Vcc - Vbias}{Rbias1} = \frac{Vbias}{Rbias2} + I_{NonInv}$$

Lastly, we combine all of them in a "**Solve**" block, and let the "Symbolics" processor grind-out an answer:

$$SysRes := \begin{pmatrix} Eqn_{1a} \\ Eqn_{1b} \\ Eqn_{1c} \\ Eqn_{2a} \\ Eqn_{2b} \\ Eqn_{3a} \\ Eqn_{3b} \\ Eqn_{3c} \\ Eqn_{4a} \\ Eqn_{4b} \\ Eqn_{4c} \\ Eqn_{4d} \end{pmatrix} \text{solve}, \begin{pmatrix} OUT \\ Vout \\ Vbias \\ Vin \\ X_{IN} \\ X_{FB} \\ X_{Cin} \\ X_{Cfb} \\ X_{Cout} \\ I_{Out} \\ I_{Inv} \\ I_{NonInv} \end{pmatrix} \rightarrow$$

Because of the large number of terms and intermediary values, the results (contained in **SysRes**) are too large to display directly. As you can see above, Mathcad doesn't even try to display the results; however, the results ARE in the SysRes variable. In fact, even stripping-apart **SysRes** to obtain the single result we care about (**OUT**), we get a long, ugly 'mess' of an equation that we can't fully reproduce here (but is visible in Mathcad by making use of the horizontal scroll bar); the beginning of it looks like:

$$OUT := \left(SysRes^{\langle 1 \rangle}\right)_1 \rightarrow \frac{-1}{2} \cdot \frac{2 \cdot i \cdot Rbias2 \cdot Vcc + i \cdot Rfb \cdot Rbias2 \cdot Iio - 4 \cdot i \cdot \pi^2 \cdot Frq^2 \cdot Cin \cdot Rin \cdot Rbias1 \cdot Rbias2 \cdot Iio \cdot Rfb \cdot Cfb + 8}$$

But, it doesn't really matter that we cannot easily READ the equation, since we are going to assign it to functions that we can call numerically. The two functions we wish to create are "**GAIN**" and "**OFFSET**", as those are the two characteristics we will later optimize the circuit around. We start-out by deriving equations for "**GAIN**" and "**OFFSET**" based on the full

"**OUT**" equation; we do this by using judicious substitutions as shown below (again, the final equations are too long to see here in their entirety, but are visible within Mathcad):

$$OFFSET := OUT \left| \begin{array}{l} \text{substitute , IN = 0V, Frq = 0Hz} \\ \text{collect , Vcc , Vio , Iib, Iio, Rfb} \end{array} \right. \rightarrow -i\cdot\frac{Rbias2}{-i\cdot Rbias1 - i\cdot Rbias2}\cdot Vcc - \frac{1}{2}\cdot\frac{-2\cdot i\cdot Rbias2 - 2\cdot i\cdot Rbias1}{-i\cdot Rbias1 - i\cdot Rbias2}\cdot Vio + \left(\frac{-1}{2}\cdot\frac{2}{}\right.$$

$$GAIN := \frac{OUT}{IN} \left| \begin{array}{l} \text{substitute , Vcc = 0V, Vio = 0V, Iio = 0A, Iib = 0A} \\ \text{collect , IN, Frq, } \pi\text{, Cout , Cfb, Cin} \end{array} \right. \rightarrow \frac{-1}{2}\cdot(4\cdot Rfb\cdot Rbias2 + 4\cdot Rfb\cdot Rbias1)\cdot Cin\cdot\pi\cdot\frac{}{(-8\cdot Rout\cdot R}$$

Notice that for deriving "**OFFSET**" we set the input signal ("**IN**") to zero and we set the frequency ("**Frq**") to zero; this is a DC calculation with the input grounded. Similarly, when deriving "**GAIN**" we set the power supply (**Vcc**) to zero, as we also do with all of the op-amp input characteristics (**Vio**, **Iio**, and **Iib**); this is an AC calculation and DC terms need to drop out. In both cases terms have been "collected" in order to obtain more readable results; this allows observation of the contributions of key parameters of the calculation (e.g. how **Vcc** and **Vio** affect the offset voltage of the circuit). Oddly enough, Mathcad forces the derivation of "**GAIN**" to have "**IN**" as one of the collection terms even though "**IN**" never actually appears in the resultant equation; for whatever reason, without that collection term you will not receive a result.

Now that these two equations have been generated, we can assign them to functions that can be called numerically (again, you cannot see the entire equation here):

$$Offset(Rfb, Rbias1, Rbias2, Vcc, Vio, Iio, Iib) := OFFSET \rightarrow -i\cdot\frac{Rbias2}{-i\cdot Rbias1 - i\cdot Rbias2}\cdot Vcc - \frac{1}{2}\cdot\frac{-2\cdot i\cdot Rbias2 - 2\cdot i\cdot Rbias1}{-i\cdot Rbias1 - i\cdot Rbias2}\cdot Vio +$$

$$Gain(Frq, Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2) := GAIN \rightarrow \frac{-1}{2}\cdot(4\cdot Rfb\cdot Rbias2 + 4\cdot Rfb\cdot Rbias1)\cdot Cin\cdot\pi\cdot\frac{}{(-8\cdot Rout\cdot Rin}$$

$$Gain_{dB}(Frq, Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2) := 20\log(\left|Gain(Frq, Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2\right.$$

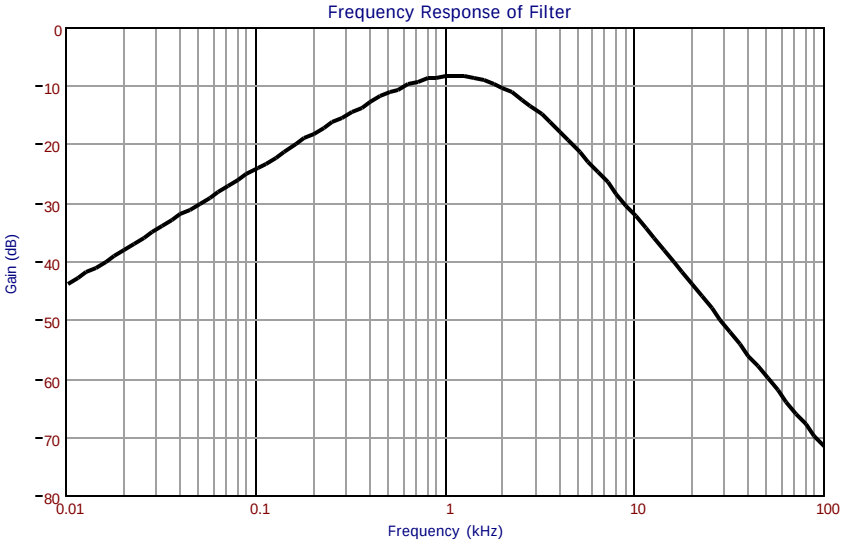
If we plug-in some initial values for the components (setting all resistors to 10kΩ and all capacitors to 0.01μF), we can quickly determine the output offset voltage and plot the Bode response of the circuit:

$Rin := 10k\Omega$ $Rfb := Rin$ $Rout := Rin$ $Rbias1 := Rin$ $Rbias2 := Rin$

$Cin := 0.01\mu F$ $Cfb := Cin$ $Cout := Cin$

$Vcc := 5V$ $Vio := 10mV$ $Iio := 1\mu A$ $Iib := 0.1\mu A$

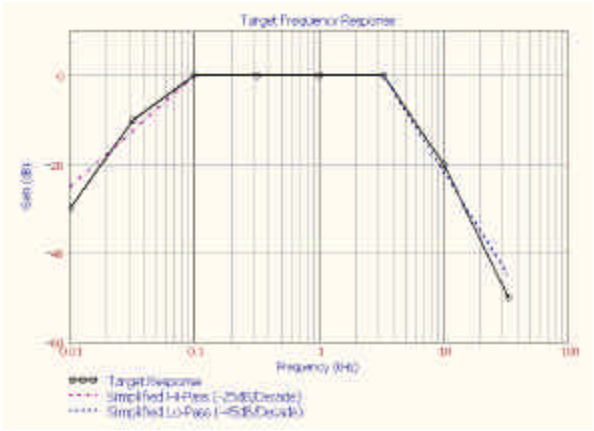
$Offset(Rfb, Rbias1, Rbias2, Vcc, Vio, Iio, Iib) = 2.4980V$



Next, we wish to optimize the circuit so that it performs to some desired response. This desired response (aka goal function) might be taken from a customer requirements document, or from a specification created by the system designer. For this example, we are going to fill a pair of vectors with Frequency/dB target points that we would like our filter's response to pass through. We can make use of an embedded Excel table to make the data-entry easy and professional looking:

$$\begin{pmatrix} Target_{Freqs} \\ Target_{dBs} \end{pmatrix} :=$$

Frequency (Hz)	Gain (dB)
10	-30
32	-10
100	0
320	0
1000	0
3300	0
10000	-20
33000	-50



As can be deduced by inspecting the table, we wish our filter to be "flat" from 100 to 3300 Hz, and have approximately 25dB/decade below and 45db/decade above that range. Obviously, real world filters have slopes of 20 and 40 dB/decade, so, our optimized filter will be "close, but no gold cigar". There is NO way to exactly provide this shape, but, perhaps we can create a filter that is "good enough".

Before we can perform our optimization, we need to define an "Error" function, which the optimizer will attempt to force to zero. For this example we will use a simple "Sum of the squared errors" algorithm. This algorithm simplistically sums the square of the errors in gain (dBs) at each of the target frequencies. The optimizer will call this function for each permutation of component values that is attempted; when the optimizer finds a combination that yields a minimal output from this function, it terminates and returns the optimal values.

$$ERROR_{dBs}(Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2) := \sum_{n=ORIGIN}^{last(Target_{Freqs})} \left(Gain_{dB}\left(Target_{Freqs_n}, Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2\right) - Target_{dBs_n}\right)^2$$

Finally we use the "**Given…Minerr**" construct to optimize the circuit element values:

Given

$ERROR_{dBs}(Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2) = 0$

$Offset(Rfb, Rbias1, Rbias2, Vcc, Vio, Iio, Iib) = 2.5000V$

$1k\Omega \le Rin \le 100k\Omega$ $\quad 1k\Omega \le Rfb \le 100k\Omega$ $\quad 1k\Omega \le Rout \le 100k\Omega$

$100pF \le Cin \le 1\mu F$ $\quad 100pF \le Cfb \le 1\mu F$ $\quad 100pF \le Cout \le 1\mu F$

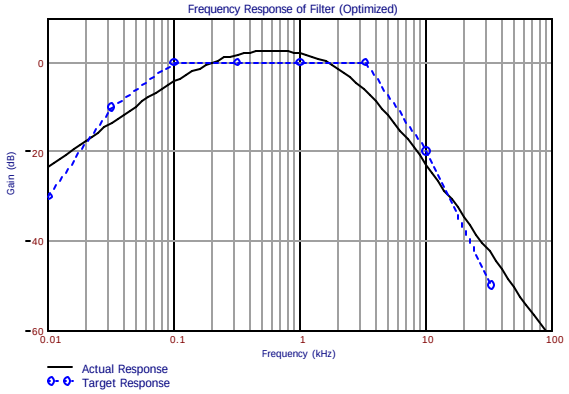
$$\begin{pmatrix} Rin \\ Cin \\ Rfb \\ Cfb \\ Rout \\ Cout \\ Rbias1 \\ Rbias2 \end{pmatrix} := Minerr(Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2)$$

The component values that were chosen are:

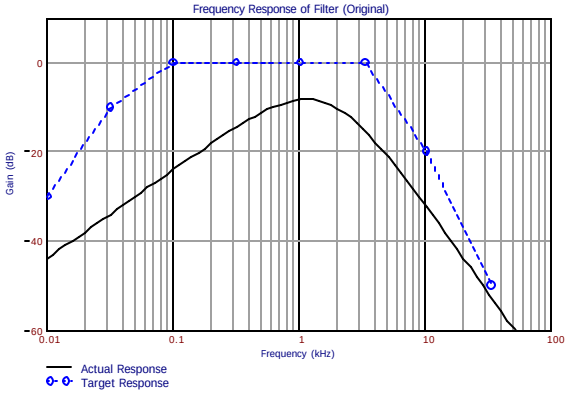
Rbias1 = 9.9029kΩ	Rbias2 = 9.9018kΩ		
Rin = 8.6275kΩ	Rfb = 13.5240kΩ	Rout = 26.9265kΩ	V_{Offset} = 2.5000V
Cin = 0.0780µF	Cfb = 0.0054µF	Cout = 0.0027µF	$Gain_{1kHz}$ = 2.0310
Cin = 7.8032×10^4pF	Cfb = 5358.4219pF	Cout = 2692.6482pF	

We can plot our initial response and our optimized response as:

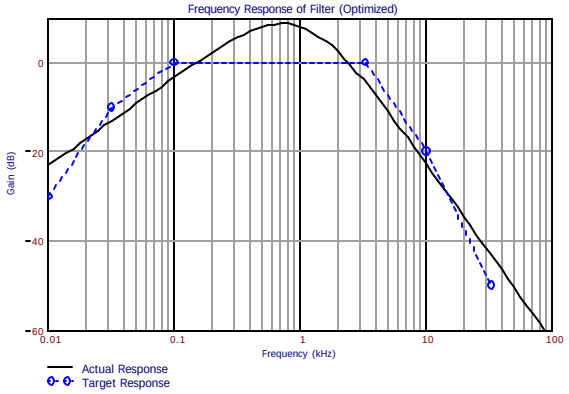
Optimized Design:



Initial Guess:



Note that we needed to specify several points within the passband in order to force Mathcad's optimizer to keep the mid-band gain down. Without these extra points the "skirts" become better matched and the mid-band gain became too large, as shown below:



In fact, with two minor modifications we can provide additional constraints to make some target points more important than others:

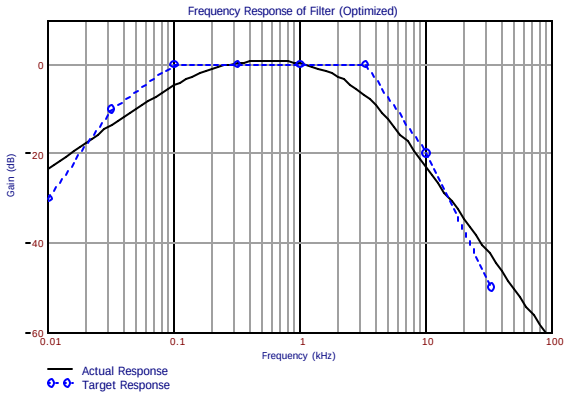
- Add a "Weighting Factor" column to the data-entry table (see following examples)
- Modify the "Error" function to multiply the squared error by the weight

$$ERROR_{dBs}(Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2) := \sum_{n=ORIGIN}^{last(Target_{Freqs})} \left[Weights_n \cdot \left(Gain_{dB}\left(Target_{Freqs_n}, Rin, Cin, Rfb, Cfb, Rout, Cout, Rbias1, Rbias2\right) - Target_{dBs_n} \right)^2 \right]$$

Specifying that the two mid-band targets are 10X more important than the others yields the following optimization results:

$$\begin{pmatrix} Target_{Freqs} \\ Target_{dBs} \\ Weights \end{pmatrix} :=$$

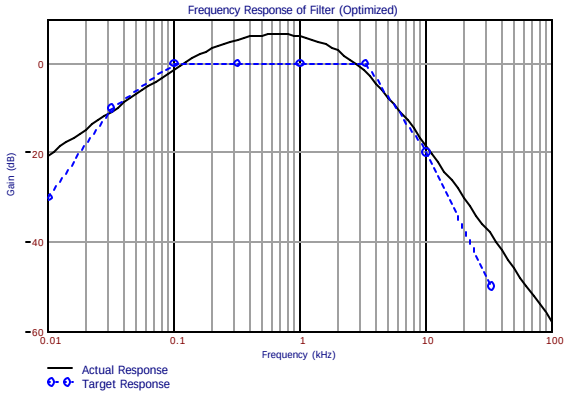
Frequency (Hz)	Gain (dB)	Weight
10	-30	1
32	-10	1
100	0	1
320	0	10
1000	0	10
3300	0	1
10000	-20	1
33000	-50	1



Specifying that the two corner frequency targets are 10X more important than the others yields the following optimization results:

$$\begin{pmatrix} Target_{Freqs} \\ Target_{dBs} \\ Weights \end{pmatrix} :=$$

Frequency (Hz)	Gain (dB)	Weight
10	-30	1
32	-10	1
100	0	10
320	0	1
1000	0	1
3300	0	10
10000	-20	1
33000	-50	1



Conclusion

Mathcad is a very fast and powerful mathematical analysis package that can be used by electrical designers to perform a variety of design and analysis tasks ranging from simple to complex. The built-in "symbolic processor" allows us to easily construct complex transfer functions from a simple collection of nodal equations. The built-in "optimizer" allows us to easily arrive at optimal component values such that the circuit meets a given set of constraints (targets). With these two features alone (and Mathcad has many more) EEs can perform better, more accurate circuit designs than pen-and-paper methods; in many instances better, more efficient circuit designs than using circuit simulators (like SPICE).

Biography:

Jim Bach received a BSEE (with a minor in Computer Science) from Marquette University in 1982. He began working for Delphi Electronic & Safety (DES) in 1986, when it was called Delco Electronics. In his career at DES he has been a Systems Engineer, an Advanced Development Engineer, an EE Simulation and Modeling Engineer, and now EE Analysis Engineer and Mathcad Instructor. His primary background is in Powertrain Electronics (engine and/or transmission control modules), although he's assisted engineers in other product lines. Primarily "analog" in nature, he enjoys working with circuits that interface with sensors or control solenoids, as well as conditioning/filtering signals. Over the past few years Jim has become DES's resident expert in utilizing Mathcad for performing design and analysis tasks, to the point of having created an internal 6-day, 8-session "Mathcad for Engineers" training class and hosting periodic "Brown-Bag Lunch" seminars. Jim enjoys circuit design and analysis, and the analytical tool known as Mathcad; putting the two of them together and teaching about it is both challenging and rewarding.

Abstract

One of Mathcad's strong points is that it has a built-in "symbolic processor", which can be used to combine a collection of nodal (circuit) equations and synthesize a set of equations (e.g. transfer functions) for the circuit. This provides a simple and automated method of "solving for N unknowns from N equations". The article explains how to use this feature to create transfer functions for a simple circuit; later the article illustrates how to use this feature to create the transfer function of a more complicated circuit (bandpass filter) and create a numeric function that can be used for design optimization and analysis.

Another of Mathcad's strong points is that it has a built-in "optimizer" capability, which can be used to adjust any number of system variables until a set of "goal" conditions have been met, or met with minimal error. This allows the engineer to make some initial guesses for component values, and then have Mathcad figure-out what the optimal values would be (say, to obtain a desired frequency response in a multi-stage filter network). The designer can constrain component values to particular ranges, to prevent the optimizer from finding problematically "too small" or "too large" of values. The designer can also create his own "Error" function, which can be used to control the weighting of trade-offs in the optimization process. This article explains how to use the optimizer to determine component values of a simple circuit in order to make it meet some criteria; later the article illustrates how to use this feature to optimize a more complicated circuit (bandpass filter) so that it approximates a desired Bode response.