Optimal Filter Design: The Simplified Real Frequency Technique

John Rinehart
Department of Physics and Astronomy, University of Waterloo

December 17, 2015

Contents

Motivation **1**

Theory **2**
Optimization of Transducer Gain 2
Kramers-Kronig Relations 2
The Gewertz Method 3
Darlington Sythesis 4

Filter Example **5**

Future Work **7**

Conclusion **7**

Appendices **8**

A MATLAB® Code **8**
A.1 Example9_2_1.m 8
A.2 CheckDecreasing.m 10
A.3 ErrorFunction.m 11
A.4 Gewertz.m 11
A.5 PoleExtractionAtInfinity.m 13
A.6 RsFromDs.m 14
A.7 TransducerGain.m 14
A.8 XsFromDs.m 15

Introduction

Filters are an important part of the RF/Microwave signal chain. They can be used to prevent aliasing of signals used with nonlinear devices (like mixers) and also to prevent interference from signals with proximal frequency components. Ever filter is qualified by a number of metrics: insertion loss, rejection, selectivity, group delay, etc. The topology, number and quality of components determines the filter's behavior and, as such, must be chosen very carefully.

To add to the filter designer's struggles, it can be shown that it is not possible to design a filter with arbitrarily large bandwidth and arbitrarily high gain. The derivation of the gain-bandwidth theorem can be found in reference [1]. Thus, it is up to the designer to push the gain-bandwidth limits to the limit. This requires high-fidelity components and an optimal design scheme. Material science and component design is under active research. This will improve the reliability and performance of the components, themselves.

What is needed, then, is to consider a methodology with which the proper components for an optimal filter design can be considered. The simplified real frequency technique (SRFT), devised in 1982 by Herbert Carlin and Binboga Yarman addresses this problem [2]. Essentially, the SRFT achieves an optimal filter design by using computer-aided optimization. This report will describe SRFT and a MATLAB® implementation of the SRFT currently under development. The goal of this report is to demonstrate the efficacy of the SRFT and a particular microwave design which was implement using the aformentioned MATLAB® implementation.

Theory

The theory behind the SRFT is simple and relies on 4 basic ideas:

1. Optimization of Transducer Gain
2. Kramers-Kronig Relations
3. The Gewertz Method
4. Darlington Synthesis (Pole Extraction at Infinity)

Optimization of Transducer Gain

The concept of transducer gain optimization is simple: Consider the source and the load and allow the introduction of some network that maximizes the power transfer from the source to the lead. This power transfer will be maximum when the source and the load are conjugately matched. Thus, the problem can be stated as follows:

$$\min_{\text{networks}} ||T_{\text{goal}}(\omega) - T_{\text{network}}(\omega)||$$

The idea, stated above, is to declare some goal transducer gain function (that may be physically impossible based on gain-bandwidth considerations). Then, considering all possible networks, we minimize over the norm of the difference between the two transducer gain functions. However, we need some way in which to perform this minimization. So, we next express the transducer gain of the network in terms of the impedance of the network and the load:

$$T(\omega) = \frac{4rR}{(x+X)^2 + (r+R)^2}$$

Where, above, lower case r and x indicate load quantities and R and X represent network quantities. It should be clear from this description that the frequency dependence is captured by all of the four circuit quantities: r, R, x, X. It is precisely this frequency dependence that determines the transducer gain. It should also be noted that the gain appears to be maximized when the reactance of the load is cancelled by the reactance of the introduced network. This is a restatement of the conjugate matching principle stated earlier. Now, for the purposes of this report, the load will always be considered fixed. The network is introduce in the context of the load and the transducer gain is maximized.

They way in which SRFT attempts to determine the optimum resistance characteristic is determined by the relationship between the resistance and reactance characteristic and will be discussed subsequently.

Kramers-Kronig Relations

A really interesting side-effect of having access to a linear, causal system is that the real and imaginary parts of the transfer function of such a system are not independent. It can be shown that the relationship between the real and imaginary parts of a circuit network R and X are as follows (see [1]):

$$R(\omega) = -\frac{2}{\pi}\int_0^\infty \frac{\Omega\ X(\Omega)}{\Omega^2-\omega^2}d\Omega$$
$$X(\omega) = \frac{2\omega}{\pi}\int_0^\infty \frac{R(\Omega)}{\Omega^2-\omega^2}d\Omega$$

These are known as the Kramers-Kronig relations. It is at this point that the numerical optimization procedure is introduced. Imagine that the resistance as a function of frequency is broken up into many piecewise segments. Then, for each break frequency ω_k there exists some resistance R_k at that break frequency. $R(\omega)$ can now be expressed as:

$$R(\omega) = R_0 + \sum_{k=1}^{n} D_k a_k(\omega)$$

where

$$a_k(\omega) = \begin{cases} 1 & \omega \geq \omega_k \\ \frac{\omega-\omega_{k-1}}{\omega_k-\omega_{k-1}} & \omega_{k-1} \leq \omega \leq \omega_k \\ 0 & \omega < \omega_{k-1} \end{cases}$$

It is clear from this, then, that D_k represents the slope of the kth break frequency and, as such:

$$D_k = R_k - R_{k-1}$$

If this is the form of $R(\omega)$ that is assumed for the SRFT then it can be shown that the reactance must be related to $R(\omega)$ by:

$$X(\omega) = \sum_{k=1}^{n} D_k b_k(\omega)$$

where

$$\begin{aligned} b_k(\omega) = (\pi(\omega-\omega_{k-1}))^{-1}\big(\, & (\omega+\omega_k)\ln|\omega+\omega_k| + \\ & (\omega-\omega_k)\ln|\omega-\omega_k| - \\ & (\omega+\omega_{k-1})\ln|\omega+\omega_{k-1}| - \\ & (\omega-\omega_{k-1})\ln|\omega-\omega_{k-1}|\,\big) \end{aligned}$$

This expression for $X(\omega)$ is nothing more than the previous expressions relating $R(\omega)$ to $X(\omega)$ assuming that $R(\omega)$ has a piecewise linear form. This piecewise linearity makes it easy to calculate the reactance on a computer. Since the goal is to reduce the difference between the goal gain function and the network gain function, the algorithm will perform the following: It will adjust the D_ks using some scheme (gradient ascent, random, etc.) and will attempt to produce a set of D_ks that makes the difference between $T_{network}$ and T_{goal} a minimum. Those D_ks will correspond to some $R(\omega)$ and some $X(\omega)$. Thus, it would seem that the D_k correspond to some network. It is this point that we have to introduce the way in which the network is deduced from $R(\omega)$ and $X(\omega)$.

The Gewertz Method

In order to determine a network from the obtained piecewise linear $R(\omega)$, I can perform the following. It can be shown that the impedance of the network and the resistance of the network are related:

$$R(-s^2) = \frac{1}{2}\left(Z(s) + Z(-s)\right)$$

where $s = j\omega$. The reason for expression R as a function of s^2 is that the resistance function must be even about $\omega = 0$, by physical considerations. So, now, consider that the piecewise linear $R(\omega)$ is approximated by some rational function fit $\tilde{R}(-s^2) = \frac{p(-s^2)}{q(-s^2)}$. Now, consider $Z(s) = \frac{n(s)}{d(s)}$, where n and d are the numerator and denominator polynomials of $Z(s)$, respectively. It can be shown, then, that by relating the coefficients of $p(-s^2)$ with those of $d(s)$ and $n(s)$ that the following holds (note that $d(s)d(-s) = q(-s^2)$):

$$\begin{pmatrix} p_0 \\ p_2 \\ \vdots \\ p_{2m} \end{pmatrix} = \begin{pmatrix} d_0 & 0 & \dots & 0 & 0 \\ d_2 & -d_1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & -d_{m-1} & d_{m-2} \\ 0 & 0 & \dots & 0 & d_m \end{pmatrix} \begin{pmatrix} n_0 \\ n_1 \\ \vdots \\ n_{m-1} \\ n_m \end{pmatrix}$$

Above, it is assumed that $p(s) = \sum_{i=1}^{2m-2} p_i s^i$ and similarly with the other polynomials.

Now, our goal is to solve for $Z(s) = \frac{n(s)}{d(s)}$. We can obtain the coefficients of $d(s)$ from those of $q(s)$ (we have $q(-s^2)$ from our fit of $R(-s^2)$) by using spectral factorization. According to [1], any real non-negative function $A(-s^2)$ can be written as a product, $a_0^2 a(s)a(-s)$, where a_0 is real and $a(s)$ is a real monic polynomial. Thus, we know the coefficients $d_0, d_1, d_2, \dots, d_{2m}$ which form $d(s)$ since we know $q(-s^2)$ from having had fit $R(s)$ to a rational function.

We can then invert the above equation to find the numerator polynomial coefficients. Once we have $n(s)$ from the aforementioned procedure, we form $Z(s) = \frac{n(s)}{d(s)}$. This is the Gewertz method.

What we need now is a way in which to obtain the network (resistors, inductors, capacitors) from $Z(s)$. This is what Darlington synthesis enables us to do.

Darlington Synthesis

It is shown in reference [1] that any realizable impedance function can be realized as a lossless LC ladder network terminated on a resistor. The proof will not be given here but a demonstration of the technique will be described. Assuming that $Z(s)$ is expressed as a rational function (which can always be done to at least some approximation), then the network can be constructed as follows:

$$Z(s) = \frac{n_0 + n_1 s + n_2 s^2 + \dots + n_m s^m}{d_0 + d_1 s + d_2 s^2 + \dots + d_m s^m}$$

Then, by iterative division, $Z(s)$ can be expressed as a continued fraction:

$$Z(s) = \frac{1}{\frac{d_0 + d_1 s + d_2 s^2 + \dots + d_m s^m}{n_0 + n_1 s + n_2 s^2 + \dots + n_m s^m}}$$

Then, using long division, this can be expressed in the form:

$$Z(s) = \frac{1}{c_0(s) + \frac{1}{c_1(s) + \frac{1}{c_2(s) + \dots}}}$$

Now, this is exactly the form of an LC ladder network input impedance. By construction, $c_0(s)$ is at most a first order polynomial in s. This holds true for other $c_i(s)$ also. Consider the case, for example, when $c_0(s) = .5s$. This is exactly a capactive admittance of value .5s S which corresponds to a capacitance of 0.5 F. Thus, the first element in the network would be a shunt capacitance (since it's an admittance) of 0.5 F. So, apparently, expressing the impedance in this form by repeating iterative long division is a way in which the components of the network are made easily visible.

Once the components are identified, all that remains is to simulate the network determined by the algorithm to ensure that it, indeed, satisfies the properties desired by the designer; that is, that it approximates well the optimal gain-bandwidth characteristic.

Now that the theory has been established, a practical example of a filter will be undertaken and the network will by synthesized and simulated. The shortcomings of the current implementation will be discussed subsequently.

Filter Example

The following example is taken from section 9.5.2 of reference [3]. The load impedance is a series inductor of value 2.3 H, a shunt capacitor of value 1.2 F and a parallel resistor (the load resistance) of value 1 Ω. Thus, we can express $z(s) = 2.3s + 1/(\frac{1}{1.2s} + \frac{1}{1})$ (note the lower case usage of z for the load). I have used the code included in appendix A to generate the following filter design:

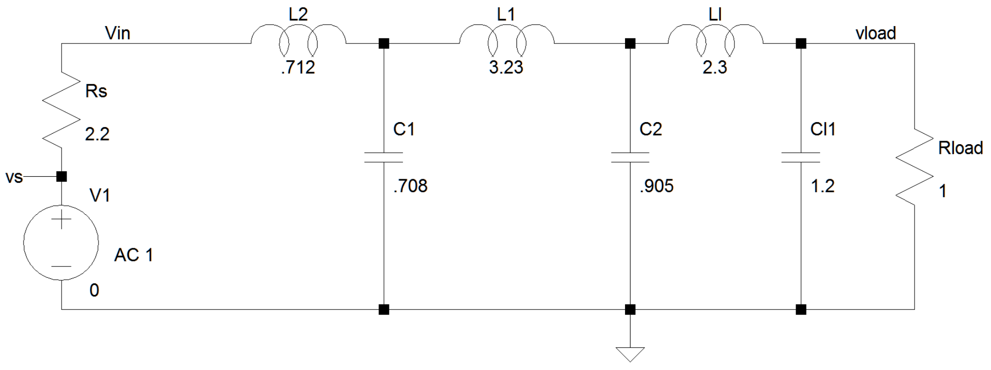


Figure 1: Schematic determined by the SFRT

The results of having had simulated this network in LTSpice® are given below:

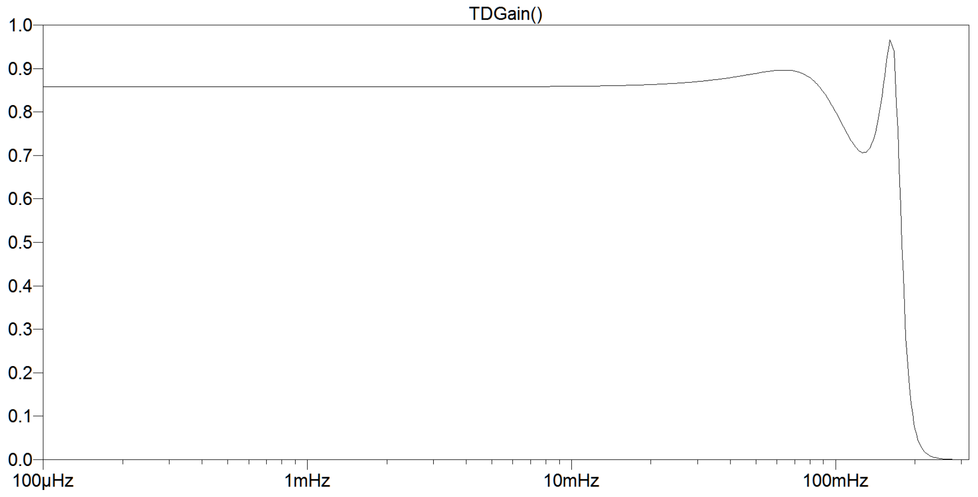
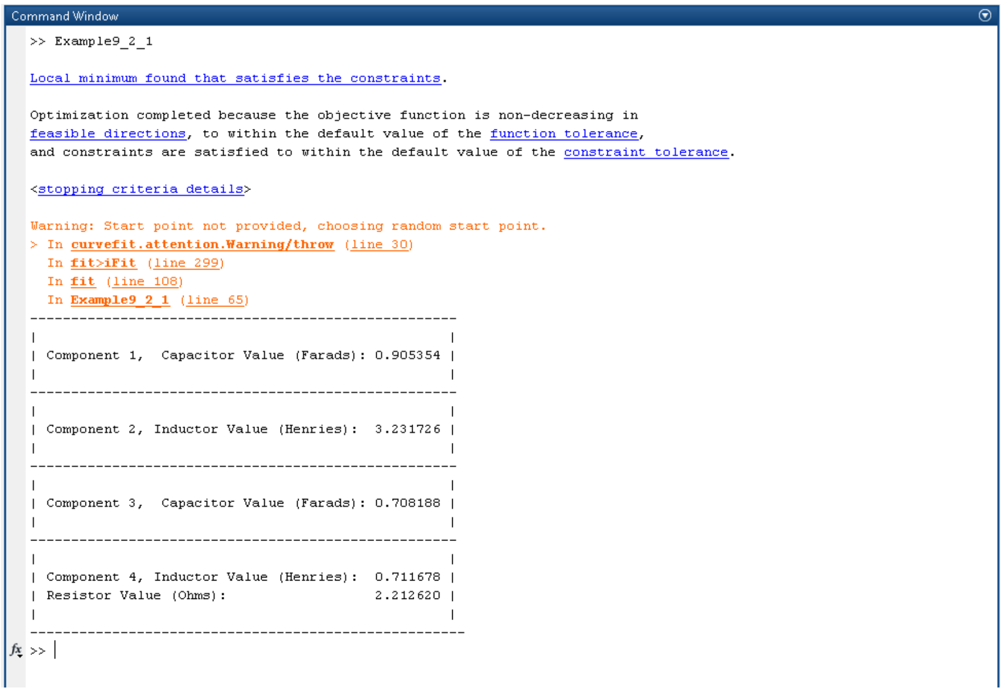


Figure 2: Results of having had simulated the network of figure 1 in LTSpice®

These component values are currently returned to the MATLAB® command line window once the entire SRFT has been completed. The command line output for this example is shown in the figure below.



```
>> Example9_2_1

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.

<stopping criteria details>

Warning: Start point not provided, choosing random start point.
> In curvefit.attention.Warning/throw (line 30)
  In fit>iFit (line 299)
  In fit (line 108)
  In Example9_2_1 (line 65)
-----------------------------------------------------
|                                                   |
| Component 1,  Capacitor Value (Farads): 0.905354 |
|                                                   |
-----------------------------------------------------
|                                                   |
| Component 2, Inductor Value (Henries):  3.231726 |
|                                                   |
-----------------------------------------------------
|                                                   |
| Component 3,  Capacitor Value (Farads): 0.708188 |
|                                                   |
-----------------------------------------------------
|                                                   |
| Component 4, Inductor Value (Henries):  0.711678 |
| Resistor Value (Ohms):                  2.212620 |
|                                                   |
------------------------------------------------------
>> |
```

Figure 3: Output of the SRFT algorithm in the MATLAB® command line window

This simulation can be compared to the goal and obtained transducer gain (shown below):

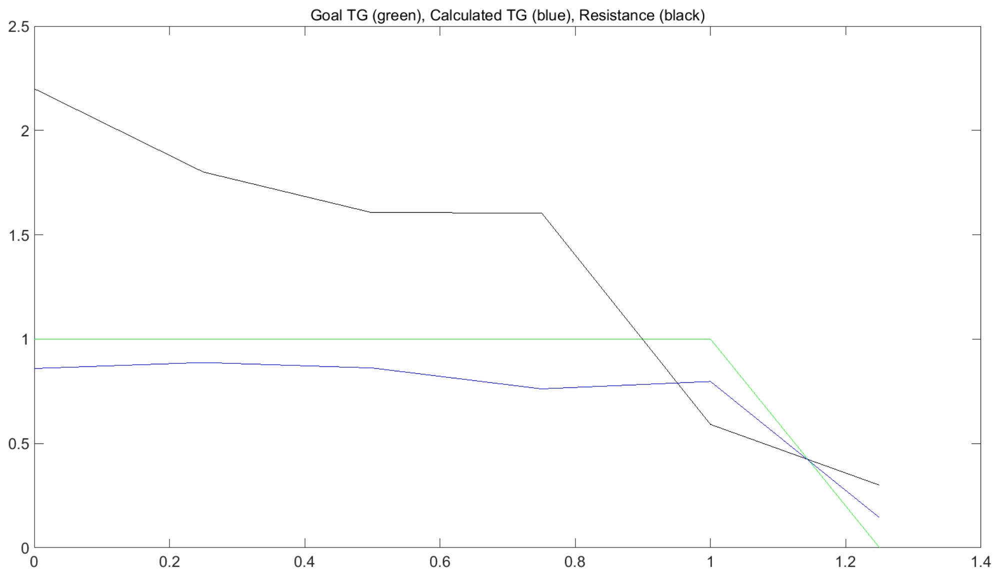


Figure 4: MATLAB® calculated piecewise resistance (black), optimum/goal gain function (green) and numerically-obtained gain function (blue). The 6 break points ($\omega = 0, .25, .5, .75, 1.0, 1.25$) are clearly visible in the piecewise resistance plot. The same vertical axis was used for both the transducer gain and the resistance plot. The resistance vertical axis has units of ohms and the transducer gain is normalized to one (power delivered to the load is the same as the available power from the source); the horizontal axis has units of rad s^{-1}

The MATLAB® results were obtained assuming 6 break frequencies. $\omega = 0, .25, .5, .75, 1.0, 1.25$. There are discrepancies that exist between the LTSpice® results and the MATLAB® results. There is good reason for this. One reason is that the MATLAB® results were obtained using component values that were precise to within the numerical precision of the calculations. A more significant reason for the difference, however, is that the MATLAB® implementation of the SRFT only calculates the transducer gain at select points between the break points. The LTSpice® results are solved over a more continuous bandwidth.

The reason for the seeming discrepancy between the horizontal axes is that the LTSpice® results are plotted against linear frequency (Hz) while the MATLAB® results are plotted against angular frequency ($\mathrm{rad\,s^{-1}}$).

So far, no microwave circuit has been designed. However, this topology lends itself well to design using open stubs on microstrip. All that would be needed would be to transform the lumped components into transmission line components using Kuroda's identities (to turn all inductors into capacitors, realizing open shunt stubs instead of series short stubs) and Richard's transformations to turn shunt capacitors into open shunt stubs of $\lambda/8$ length (45°). Time, unfortunately, did not permit for this. The code generation took quite some time and it is not yet optimal. Future improvements are discussed below.

Future Work

There are a number of problems with the code base that was used to replicate the circuit studied in reference [3] (this circuit was also used as an example in [1]). The first main problem with the code is the reliance on fmincon , a MATLAB® function that is able to solve nonlinear optimization problems with applied constraints. In this case, the constraint of decreasing resistance with increasing frequency was applied.

However, nonlinear optimization is subject to the unfortunate problem that solutions can arise that are not, in general, the best solution. The optimization procedures, in general, work by determining the best way in which to "tweak"/adjust the variables under consideration such as to minimize the function at hand. The solution relies, then, heavily on the initial conditions given to the solver. A large improvement to the algorithm would be to express the problem as that of a linear optimization problem. Then, the sensitivity to initial conditions is gone and a global minimum is guaranteed.

Next, the Gewertz method that is used in conjunction with Darlington Synthesis is undesirable. The amount of steps that are undertaken to obtain $Z(s)$ from $R(-s^2)$ are so many that the chance for numerical or implementation error are great. A better way to approach the problem would be to obtain $X(s)$ directly from the piecewise $\tilde{R}(s)$ obtained from the optimization procedure. Then, once $X(s)$ is known it is easy to construct $Z(s) = R(s) + X(s)$. Then, a rational function fit of $Z(s)$ can be obtained (to arbitrary numerator and denominator precision).

Furthermore, this tool, right now, does not accommodate the "double matching" problem where both the source and the load are fixed and the best network is to be obtained. In this tool, the load is fixed and the rest of the network is allowed to vary.

Finally, a great edition to this tool would be to increase the level of interactivity and usability. Right now, the script Example9_2_1.m uses the supplied functions in a very particular way that make it easy to adjust the load, the number of breakpoints, etc. However, it should be possible to ship the entire SRFT library without the dependence on such a particular script.

Conclusion

In summary, a case study from the literature was studied in an attempt to write a SRFT tool in MATLAB®. A beta version of such a tool was produced and is under active revision. Currently, this code is available at `https://github.com/fuzzybear3965/SRFT`. A fork of this code base will be written in Julia and in Java. This will make the adoption of such a tool much easier. The SRFT allows for optimal filter design which utilizes components to the best degree possible and attains on optimal design, which is too often difficult to achieve elsewhere in microwave engineering.

One further note, the entire assumption of the SRFT is that the system is linear. Based on this assumption, the principles of elementary circuit theory and the relations between the real and imaginary parts of the transfer function could be used to obtain the results given. However, in many cases, the circuit or system under consideration behaves nonlinearly. In this case, the theory given is no longer applicable. Another goal of this project, then, is to extend the usefulness of SRFT into the domain of nonlinear signals. That has great applicability to the design world and should be of great interest to industry and to academia.

Appendices

A MATLAB® Code

Below, I have included all of the source code necessary to replicate my results. The code base is open to many revisions (as discussed in the report). I plan on making this SRFT project open to the public on Github. This way, SRFT design can be open to the public.

Nota bene: The code is well-documented. However, there are many blocks that are commented out that were for testing purposes that can still be used for that purpose. In my opinion, these blocks do not make reading the source inordinately difficult.

A.1 Example9_2_1.m

The below file is the script file which uses all of the other function files that are included below.

```
clear;
% number of breakpoints and fit order below
numbrkpts = 6;
%fitorder = 6; % make sure it's even or things will break
% load description below
zload = @(w)((1+1i*w*1.2).^-1+1i*w*2.3); % g = 1, Yc = 1i*w*1.2, Zl = i*w*2.3
% Break frequencies below
omega_max = 1.25;
break_omegas = linspace(0,omega_max,numbrkpts)';
% Frequencies over which to fit the polynomial
%optim_freqs = (0:1/20:omega_max+1)';
optim_freqs = break_omegas;
omega_one_idx = find(optim_freqs>1,1);
% Transducer gain below
gaingoal = ones(length(optim_freqs),1);
%gaingoal(omega_one_idx:end) = linspace(1,0,length(gaingoal)-omega_one_idx+1)
    ';
gaingoal(omega_one_idx:end) = 0*ones(length(optim_freqs)-omega_one_idx+1,1);

R0 = 2.2;
% TODO: Make an initial guess function based on reactive component
% cancellation
Dinit = [-.3; -.4; -.3; -.6; -.6];

objective = @(D)ErrorFunction(break_omegas,optim_freqs,zload,D,R0,gaingoal);
sum_constraint = -1*ones(1,length(Dinit)); % Make sure the sum is less than R0
% x = fmincon(@myfun,x0,A (inequality),b,Aeq (equality),beq,lb,ub,@mycon)
```

```
[Ds, TG] = fmincon(objective,Dinit,sum_constraint,R0-.3,[],[],[],[],
    @CheckDecreasing); %TODO: Investigate other optimization scheme
%Ds = Dinit;
R = RsFromDs(break_omegas,optim_freqs,Ds,R0);

if (any(R < 0))
    warning(['Your fit didn''t determine no resistance at the highest '...
        'frequency. Check your fit.']);
    for i = 1:length(R)
        if R(i) < 0
            R(i) = 0.1;
        end
    end
end

X = XsFromDs(break_omegas, optim_freqs, Ds); % Obtain X to check TG

z = zload(optim_freqs);
Z = R + 1i*X;
fig=figure(1);
plot(optim_freqs,gaingoal,'g',optim_freqs,R,'k',optim_freqs,TransducerGain(Z,z
    ),'b'); % Check TG
title(['Goal TG (green), Calculated TG (blue), '...
    'Resistance (black)']);

total_freqs = [-1*flipud(optim_freqs(2:end)); optim_freqs];
%normalized_R = R / R(1); % Turn the numerator into a "one".
total_R = [flipud(R(2:end)); R]; % Make the double-sided R, assuming even
%total_R = [fliplr(normalized_R(2:end)),normalized_R]; % Make the double-sided
    R, assuming even
InverseR = 1./total_R; % We assume the denominator of R is a polynomial

% Perform Polynomial Fitting
%[xData, yData] = prepareCurveData( total_omegas, InverseR );
%ft = fittype(sprintf('poly%d',fitorder));\
%[fitresult, gof] = fit( xData, yData, ft );

% Perform Custom Fitting
[xData, yData] = prepareCurveData( total_freqs, InverseR);
ft = fittype('a8*x.^8+a7*x.^7+a6*x.^6+a5*x.^5+a4*x.^4+a3*x.^3+a2*x.^2+a1*x.^1+
    a0','independent','x');
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
[fitresult, gof] = fit(xData, yData, ft, opts);
% Below: First term is made highest order, consistent with roots and poly
resistance_den_coeffs = flipud(coeffvalues(fitresult)');

% Below: Check to make sure resistance_den_coeffs are good.
figure(2);plot(optim_freqs,R,'b',optim_freqs,1./polyval(resistance_den_coeffs,
    optim_freqs),'r');
title('Polynomial fit resistance (red) and Optimized Resistance (blue)');

```

```
% Make sure that the polynomial is non-negative
%figure(3);plot(-50:.01:50,polyval(resistance_den_coeffs,-50:.01:50));
% December 5: Good up to here.

% coeffs need to be ordered from greatest to least power
[numz, denz] = Gewertz(1, resistance_den_coeffs);
[components,~] = PoleExtractionAtInfinity(numz,denz);

% Print the component values below
for i = 1:length(components)
    fprintf('-------------------------------------------------------\n')
    fprintf('|                                                     |\n')
    if i == length(components)
        if mod(i,2) == 0
            fprintf('| Component %d, Inductor Value (Henries):  %f |\n',i,
               components{i}(1))
            fprintf('| Resistor Value (Ohms):                   %f |\n',
               components{i}(2))
            fprintf('|                                                     |\n')
            fprintf('-------------------------------------------------------\n')
        else
            fprintf('| Component %d, Capacitor Value (Farads):  %f |\n',i,
               components{i}(1))
            fprintf('| Resistor Value (Ohms):                   %f |\n', 1/
               components{i}(2));
            fprintf('|                                                     |\n')
            fprintf('-------------------------------------------------------\n')
        end
    elseif mod(i,2) == 0
        fprintf('| Component %d, Inductor Value (Henries):  %f |\n',i,
           components{i}(1))
        fprintf('|                                                     |\n')
    elseif mod(i,2) ~= 0
        fprintf('| Component %d,  Capacitor Value (Farads): %f |\n',i,
           components{i}(1));
        fprintf('|                                                     |\n')
    end
end
```

A.2 CheckDecreasing.m

```
function [ineq, eq] = CheckDecreasing(item)
% This function is used to ensure the D values are not increasing
if isrow(item)
    item = item';
end
ineq = item; % fmincon will try to keey all of ineq <= 0... so, we're good
%ineq = 0;
ineq(end) = 0; % The end won't make sense
eq = 0;
```

A.3 ErrorFunction.m

```
function ErrorValue = ErrorFunction(break_freqs, function_freqs, loadfunc, Ds,
    Rdc, gaingoal)
R = RsFromDs(break_freqs, function_freqs, Ds, Rdc);
X = XsFromDs(break_freqs, function_freqs, Ds);
Z = R + 1i*X;
z = loadfunc(function_freqs);
TG = TransducerGain(Z,z);
ErrorValue = gaingoal-TG;
ErrorValue = sum(ErrorValue.^2);
```

A.4 Gewertz.m

```
function [num,den] = Gewertz(R0, resistance_denominator_coefficients)
%%% Gewertz(res_num_coeff,res_den_coeff) takes two arguments:
%%% 1) Coefficients of the numerator of the resistance polynomial R. These
%%% must be specified in array where the coefficents are ordered greatest
%%% to least.
%%% 2) Coefficients of the denominator of the resistance polynomial R.
%%% These must be specified in the same way as the numerator.
%%% e.g. R(w^2) = R(-s^2) = (2.2*w^2+1)/(1+2*w^2+3*w^4-5*w^6)
%%% = (-2.2*s^2+1)/(1-2*s^2+3*s^4+5*s^6).
%%% res_num_coeff = [-2.2 0 1]; res_denom_coeff = [5, 0, 3, 0, -2, 0, 1]
% The resistance_numerator_cofficients are the coefficients of the
% numerator polynomial of r(-s^2), the resistance_denominator coefficients
% are the coefficients of the denominator polynomial of r(-s^2). These
% coefficients must adhere to the MATLAB standard of ordering coefficients
% from greatest power to least power
% If the number of numerator coeffecients is n, the number of denominator
% coefficients should be 2n-1 (i.e., numerator is 3rd order
%(4 coefficients), denominator is 6th order (7 coefficients)).

resistance_denominator_coefficients = wCoeffsTosCoeffs(
    resistance_denominator_coefficients);
ResistanceSpectralRoots = roots(resistance_denominator_coefficients); % Find
    all the roots of the denominator
ResistanceSpectralRoots = RemoveSmallRealPart(ResistanceSpectralRoots);
NegSpectralRoots = BuildNegPolyArray(ResistanceSpectralRoots); % Remove the
    zero entries corresponding to RHP roots

if isrow(poly(NegSpectralRoots))
    NegSpectralPoly = poly(NegSpectralRoots)'; % PosSpectralDen is the vector
        of polynomial coefficients
end
NegSpectralPoly = sqrt(resistance_denominator_coefficients(1))*NegSpectralPoly
    ;
ds = NegSpectralPoly;

if length(NegSpectralRoots) ~= length(ResistanceSpectralRoots)/2
    disp('Bad spectral factorization');
end
```

```

% Form the matrix of d coefficients
D = zeros(length(NegSpectralPoly),length(NegSpectralPoly));
for i = 1:length(NegSpectralPoly) % row
    for j = 1:length(NegSpectralPoly) % column
        Dval = (2*i-1)-(j-1);
        if Dval > 0 && Dval <= length(ds)
            D(i,j) = ds(Dval);
        end
    end
end

SignFlipper = eye(size(D));
for i = 1 : length(D)
    SignFlipper(i,i) = -1*(1i)^(i+i);
end

D = D*SignFlipper;

ResistanceNumeratorPolynomial = zeros(1,length(D));
if isrow(ResistanceNumeratorPolynomial)
    ResistanceNumeratorPolynomial = ResistanceNumeratorPolynomial';
end
ResistanceNumeratorPolynomial(end) = R0;
num = D\ResistanceNumeratorPolynomial; %Conforms to MATLAB standard with
    highest coeff on top
den = NegSpectralPoly; % Conform to MATLAB standard, put highest coeff on top

%%%% Test code
num_star = zeros(length(num),1);
den_star = zeros(length(den),1);
for i = 1:length(num)
    num_star(end-(i-1)) = (-1)^(i-1)*num(end-(i-1));
end
for i = 1:length(den)
    den_star(end-(i-1)) = (-1)^(i-1)*den(end-(i-1));
end
reconstr_num = .5*(conv(num,den_star)+conv(num_star,den));
reconstr_den = conv(den,den_star);

end

function NewArray = RemoveSmallRealPart(OrigArray)
NewArray = size(OrigArray);
for i = 1:length(OrigArray)
    value = abs(real(OrigArray(i))/abs(OrigArray(i)));
    if value < 1e-7
        NewArray(i) = 1i*imag(OrigArray(i));
    else
        NewArray(i) = OrigArray(i);
    end
```

```
end
end

function NegPolyArray = BuildNegPolyArray(ResistanceRoots)
NegPolyArray = zeros(length(ResistanceRoots),1);
for i = 1:length(NegPolyArray)
    if real(ResistanceRoots(i)) < 0
        NegPolyArray(i) = ResistanceRoots(i);
    elseif real(ResistanceRoots(i)) == 0 && imag(ResistanceRoots(i)) < 0
        NegPolyArray(i) = ResistanceRoots(i);
    end
end
NegPolyArray(NegPolyArray == 0) = [];
end

function sCoeffs = wCoeffsTosCoeffs(Coeffs)
% This function converts coefficients of a function of w^2 to coefficients
% of a function of (-s)^2 so that spectral decomposition makes sense. It
% assumes that Coeffs(end) is the 0th order term.

sCoeffs = zeros(size(Coeffs));
for i = 1:length(Coeffs)
    power = i-1;
    powerIdx = length(Coeffs)-(i-1);
    if mod(power-2,4) == 0 % is it the 3rd term (second order) or 7th term (
       sixth order)
        sCoeffs(powerIdx) = -1*Coeffs(powerIdx);
    else
        sCoeffs(powerIdx) = Coeffs(powerIdx);
    end
end
end
```

A.5 PoleExtractionAtInfinity.m

```
function [q, r] = PoleExtractionAtInfinity(NumPolynomial,DenomPolynomial)
NumPolynomial = RemoveSmallValues(NumPolynomial);
DenomPolynomial = RemoveSmallValues(DenomPolynomial);
NumPolynomial = RemoveLeadingZeros(NumPolynomial);
DenomPolynomial = RemoveLeadingZeros(DenomPolynomial);
el = 1;
[q{el}, r{el}] = deconv(DenomPolynomial,NumPolynomial); %Assuming den is of
    greater  or equal order than num
q{el} = RemoveLeadingZeros(q{el}); r{el} = RemoveLeadingZeros(r{el});
el = el + 1;
[q{el}, r{el}] = deconv(NumPolynomial,r{el-1});
q{el} = RemoveLeadingZeros(q{el}); r{el} = RemoveLeadingZeros(r{el});
while any(r{end})
    el = el + 1;
    r{el-2} = RemoveLeadingZeros(r{el-2});
    r{el-1} = RemoveLeadingZeros(r{el-1})    ;
    [q{el}, r{el}] = deconv(r{el-2},r{el-1});
```

```
end
end

function item = RemoveLeadingZeros(item)
zerosidx = find(item,1,'first');
item(1:zerosidx-1) = [];
end

function item = RemoveSmallValues(item)
item = item(abs(item) > 10^-8);
end
```

A.6 RsFromDs.m

```
% function R = RsFromDs(optim_freqs,Ds,R0)
% R = zeros(1,length(Ds)+1);
% R(1) = R0;
% for i = 1:length(Ds)
%     R(i+1) = R(i) + Ds(i);
% end
% end

function R = RsFromDs(break_freqs, function_freqs, Ds, R0)
R = R0*ones(length(function_freqs),1);
a = AsFromBreakFreqs(break_freqs,function_freqs);
for i = 1:length(Ds)
    R = R + Ds(i)*a{i,1}(:);
end
end

function a = AsFromBreakFreqs(break_freqs,function_freqs)
a = cell(length(break_freqs)-1,1);
for i = 1:length(break_freqs)-1
    a{i,1} = zeros(length(function_freqs),1);
    for j = 1:length(function_freqs)
        w = function_freqs(j);
        wk = break_freqs(i+1);
        wkm1 = break_freqs(i);
        if w >= wk
            a{i,1}(j) = 1;
        elseif w >=wkm1 && w < wk
            a{i,1}(j) = (w - wkm1)/(wk-wkm1);
        else
            a{i,1}(j) = 0;
        end
    end
end
end
```

A.7 TransducerGain.m

```
function TG = TransducerGain(EqualizerZ,LoadZ)
TG = 4*real(EqualizerZ).*real(LoadZ)./abs(EqualizerZ+LoadZ).^2;
```

A.8 XsFromDs.m

```
% function X = XsFromDs(break_freqs , function_freqs , Ds)
%
% F = @(w,wk)( (w+wk)*log(abs(w+wk)) + (w-wk)*log(abs(w-wk)) );
% % The following bk expressions cover the cases when w = w_k or w = w_{k-1}
% % 1) w ~= w_k or w ~= w_{k-1}
% bk = @(w,ws,widx)( (pi*(ws(widx)-ws(widx-1)))^-1*( F(w,ws(widx)) - F(w,ws(
    widx-1))) );
%
% X = zeros(1,length(Ds)+1); % Make sure X is zero before we construct it
% X(1) = 0; % bk(0,wk) = 0;
% for j = 2:length(omegas) % j is going to store the omega index
%     w = omegas(j);
%     for k = 1:length(Ds)
%         wk = omegas(k+1);
%         wkm1 = omegas(k);
%         if w == wk
%             bk1 = (pi*(wk-wkm1))^-1*( ...
%                 2*wk*log(2*wk) - ...
%                 (wk + wkm1)*log(wk + wkm1) - ...
%                 (wk-wkm1)*log(wk-wkm1) ) ;
%             X(j) = X(j) + Ds(k)*bk1;
%         elseif w == wkm1 % 4th term is zero
%             wkm2 = omegas(k-1);
%             bk2 = (pi*(wkm1-wkm2))^-1*( ...
%                 2*wkm1*log(2*wkm1) - ...
%                 (wkm1+wkm2)*log(wkm1+wkm2) - ...
%                 (wkm1-wkm2)*log(wkm1-wkm2) ) ;
%             X(j) = X(j) + Ds(k)*bk2;
%         else
%             bk4 = bk(w,omegas,k+1);
%             X(j) = X(j)+Ds(k)*bk4;
%         end
%     end
% end

function X = XsFromDs(break_freqs , function_freqs , Ds)
X = zeros(length(function_freqs),1);
bs = BsFromBreakFreqs(break_freqs , function_freqs);
for i = 1:length(Ds)
    X = X + Ds(i)*bs{i};
end
X(1)=0;
end

function b = BsFromBreakFreqs(break_freqs , function_freqs)
b = cell(length(break_freqs)-1,1);
for i = 1:length(break_freqs)-1
    b{i,1} = zeros(length(function_freqs),1);
       wk = break_freqs(i+1);
```

```
        wkm1 = break_freqs(i);
    for j = 1:length(function_freqs)
        w = function_freqs(j);
        if w ~=wk && w~=wkm1
            b{i,1}(j,1) = (pi*(wk-wkm1))^-1*( ...
                (w+wk)*log(abs(w+wk)) + ...
                (w-wk)*log(abs(w-wk)) - ...
                (w+wkm1)*log(abs(w+wkm1)) - ...
                (w-wkm1)*log(abs(w-wkm1)) );
        elseif w == wkm1
            b{i,1}(j,1) = (pi*(wk-wkm1))^-1*( ...
                (w+wk)*log(abs(w+wk)) + ...
                (w-wk)*log(abs(w-wk)) - ...
                (w+wkm1)*log(abs(w+wkm1)) );
        elseif w == wk
            b{i,1}(j,1) = (pi*(wk-wkm1))^-1*( ...
                (w+wk)*log(abs(w+wk)) - ...
                (w+wkm1)*log(abs(w+wkm1)) - ...
                (w-wkm1)*log(abs(w-wkm1)) );
        end
    end
end
end
```

References

[1] H. J. Carlin and P. P. Civalleri, *Wideband Circuit Design.* CRC Press, 1997.

[2] B. Yarman and H. Carlin, “A simplified ”real frequency” technique applied to broad-band multistage microwave amplifiers,” *Microwave Theory and Techniques, IEEE Transactions on*, vol. 30, pp. 2216–2222, Dec 1982.

[3] H. Carlin and P. Amstutz, “On optimum broad-band matching,” *Circuits and Systems, IEEE Transactions on*, vol. 28, pp. 401–405, May 1981.