# **Planned Stages**

## Stage A – Initial Design

1. Design 8th order Chebyshev filter /w fp = 3kHz, fs = 4kHz, δp = 0.1, δs = 0.01 in MATLAB and get filter poles. Also, need to ensure that filter gain isn’t too high at any given frequency. This will be a loose requirement but let’s just say to keep the output magnitude at any frequency as close to one as possible within the passband. Print out filter characteristics.
2. Using Sallen-Key stages, select components for implementing this 8th order filter implemented as a cascade of four 2nd order Sallen-Key stages. (By simulate, I mean Bode plots)
3. Simulate each stage in PSIM, export data values, import into MATLAB, and compare with designed stages.
4. ~~Stage 1~~
5. ~~Stage 2~~
6. Stage 1 + Stage 2
7. Stage 3
8. Stage 1 + Stage 2 + Stage 3
9. Stage 4
10. Stage 1 + Stage 2 + Stage 3 + Stage 4
11. Now select components that are standard values closest to the chosen ones.
12. Recompute stage poles as a result of this.
13. Repeat step a) /w original design, standardized circuit, MATLAB design.

## Stage B – Initial Lab Testing/Verification

1. Go into lab and begin breadboarding circuit.
   1. Find standard components and measure /w multimeter. Note down on paper and lay them out.
   2. Verify that the LM741 op amps work.
      1. Use quick inverting amplifier circuit /w x3 and verify operation from DC to 50kHz.
      2. Note down required decoupling caps.
   3. Select appropriate breadboard and figure out needed power supplies.
   4. With measured component values, repeat Stage A step 2a, both in simulation and on breadboard. At each point, select 5 frequencies for testing, have screenshot of expected frequency response, and print. Take pictures and videos of each result. You probably will also ask some Reddit questions along the way for certain things you’re not sure about.
   5. Clean up the board and port it out for ease of use as a ver. 0 dev board.

## Stage C - KiCAD

1. Once all the kinks are worked out, time to KiCAD this mofo.
   1. Schematic
   2. PCB Layout of dev board (components that are readily available from ECE!!)

### 04-01-2021 THURS

Start Time: (around) 6:00 PM

End Time: (around) 10:00 PM

(4h) (4h)

*(This was before I laid out the task sheet)*

* + - * Use PSIM for step-by-step circuit simulation
        1. Both regular timed sim and AC sweep
        2. AC Sweep doesn't like start frequencies that are very low (1Hz takes forever to compute through)
      * From original design, 36.5k --> 33k; 732 --> 750; 1nFs, 1uFs, 100pFs are fine --> In PSIM AC sweep, caused resonance peak to shift to around 3.2337k vs 3.1131k, eh not much
      * **Remember the Agilent 33120A Function Generator expects 50ohm output, so use half the VPP that you are going for when setting it**
      * With just the first filter and no output loading, the output waveform is a little wobbly --> need bypass caps? --> 25uF used, still seeing wobble at low frequencies tho (<200Hz)... 🡪 Ask Reddit 🡪
      * MASSIVE resonance at 1.864kHz, 1.01Vpp --> 15.7Vpp!!
      * Changed PSIM simulation components 750 --> 747.4; 33k --> 32.52k; 1uF --> 1.016uF; 100pF --> 130pF
      * Still not finding simulation AC sweep matches what is seen from physical board…

### 04-02-2021 FRI

Current Stage: Stage A steps 1 and 2

Start Time: 12:00 PM

End Time: 5:00 PM

(5h) (9h)

* + - * Laid out this documentation and the task sheet
      * Reviewed the MATLAB script I wrote up for ECE444 Applied DSP for designing a Chebyshev filter given fp, fs, δp, δs specs. Copied it over to the directory for this project and modified it to output a text file representing filter characteristics, including the poles in Hz.
      * PSIM:
        1. Learned how to export the SIMVIEW data so that I can view on MATLAB 🡪 Save As .csv 🡪 MATLAB: A = readmatrix(…) 🡪 f = A(:,1); Mag = A(:,2); 🡪 semilogx(f,Mag)🡪 I modified the filt\_check.m file to include this comparison on a magnitude Bode plot
        2. Realized evalfr() in MATLAB expects frequency as a complex number 🡨 You idiot.
      * Wrote some MATLAB scripts/functions to translate designed MATLAB filter to Sallen Key stages (to be continued…).

**%% LP Chebyshev**

**fp = 3e3; fs = 4e3;**

**wp=2\*pi\*fp; ws=2\*pi\*fs; % Specs**

**delta\_p = 0.1; delta\_s = 0.01;**

**ap = -20\*log10(1-delta\_p);**

**as = -20\*log10(delta\_s);**

**% Determine parameters K and epsilon according to specs, while also**

**% choosing a more suitable wp to give buffer at spec'd wp and ws**

**K = ceil(acosh(sqrt((10^(as/10)-1) / (10^(ap/10)-1))) / acosh(ws/wp) );**

**wp = [wp, ws/cosh(acosh(sqrt((10^(as/10)-1) / (10^(ap/10)-1))) / K)]; % Range of values for wp**

**wp = mean(wp); % Select middle value**

**epsilon = sqrt(10^(ap/10)-1);**

**% Now obtain polynomials that define the TF**

**k = 1:K;**

**H0 = (mod(K,2)==1) + (mod(K,2)==0)\*(1/sqrt(1+epsilon^2)); % For even order, H0 always is 1; for odd, it's 1/...**

**pk = -wp\*sinh(asinh(1/epsilon)/K)\*sin(pi\*(2\*k-1)/(2\*K)) + ...**

**1j\*wp\*cosh(asinh(1/epsilon)/K)\*cos(pi\*(2\*k-1)/(2\*K));**

**B = H0\*prod(-pk); A = poly(pk);**

**% Now the TF!**

**w = 0:ws\*3;**

**H = B ./ (polyval(A, 1j\*w));**

**f = w / (2\*pi);**

**plot(f, abs(H), 'LineWidth', 2);**

**delta\_p = 10^(-ap/20); delta\_s = 10^(-as/20);**

**fp = wp / (2\*pi); fs = ws / (2\*pi);**

**pgon1 = polyshape([0 fp fp 0], [0 0 delta\_p delta\_p]);**

**pgon2 = polyshape([0 fp fp 0], [1 1 2 2]);**

**pgon3 = polyshape([fs fs 3\*fs 3\*fs], [delta\_s 2 2 delta\_s]);**

**hold on;**

**plot(pgon1);**

**plot(pgon2);**

**plot(pgon3);**

**hold off;**

**grid on;**

**order\_str = ["Order: ", num2str(K)];**

**% xlabel('\omega (rad/s)'), ylabel('|H(j\omega)|'), title('Magnitude Response of Chebychev Type 1 Filter');**

**xlabel('f (Hz)'), ylabel('|H(jf)|'), title(['Magnitude Response of Chebychev Type 1 Filter', order\_str]);**

**ylim([0 1.2]);**

**%% Print stuff to text file**

**fid = fopen('Chebyshev Characteristics.txt','w');**

**fprintf(fid,'CHEBYSHEV FILTER\n\n');**

**fprintf(fid,'fp: %.0f Hz\t\t\t\tfs: %.0f Hz\n',fp,fs);**

**fprintf(fid,'delta\_p: %.2f\t\t\tdelta\_s: %.2f\n',delta\_p,delta\_s);**

**fprintf(fid,'ap: %.2f dB\t\t\t\tas: %.2f dB\n',ap,as);**

**fprintf(fid,'epsilon: %f\n\n',epsilon);**

**fprintf(fid,'Filter Order: %d\n\n',K);**

**fprintf(fid,'Poles (Hz):\n');**

**pk\_Hz = pk/(2\*pi);**

**pk\_Hz = sort\_complex\_list(pk\_Hz);**

**for i=1:length(pk\_Hz)**

**fprintf(fid,'%f +- %f\n',real(pk\_Hz(i,1)),imag(pk\_Hz(i,1)));**

**end**

**fclose(fid);**

**function [poles, wn,Q,zeta,systf] = sallenKeyCircuit(R1,R2,C1,C2)**

**% Inputs**

**% R1 --> First resistor from left**

**% R2 --> Resistor between R1 and non-inverting input of op-amp**

**% C1 --> Cap between non-inverting input and ground**

**% C2 --> Feedback cap**

**% Outputs**

**% poles --> Filter poles in Hz**

**% wn --> Filter natural frequency**

**% Q --> Quality factor**

**% zeta --> Damping factor**

**% systf --> System transfer function**

**wn = 1/sqrt(R1\*R2\*C1\*C2);**

**alpha = 0.5\*(1/C1)\*(1/R1 + 1/R2);**

**Q = wn / (2\*alpha);**

**zeta = 1/(2\*Q);**

**num = wn^2;**

**den = [1, 2\*alpha, wn^2];**

**systf = tf(num,den);**

**poles = eig(systf)/(2\*pi);**

**end**

**function [R1,R2,C1,C2] = sallenKeyComponents(C,n,Q,wn)**

**% Determine m**

**f = @(m) m\*n ./ (m.^2 + 1);**

**err = @(m) abs(f(m) - Q);**

**m = fminsearch(err,10);**

**R = 1/(wn\*C);**

**R1 = m\*R; R2 = R/m;**

**C1 = n\*C; C2 = C/n;**

**end**

**C = 10e-9; n = 100; wn = 19e3;**

**f = @(Q) sallenKeyComponents(C,n,Q,wn);**

**Qmin = 0.5; Qmax = 15;**

**Q = linspace(Qmin,Qmax,1001);**

**Y = zeros(1001,4);**

**R1=0;R2=0;C1=0;C2=0;**

**for i=1:1001**

**[R1,R2,C1,C2] = f(Q(i));**

**Y(i,:) = [R1,R2,C1,C2];**

**end**

**R1 = Y(:,1); R2 = Y(:,2);**

**subplot(2,1,1);**

**plot(Q,R1,'LineWidth',2);**

**title('R1 vs Q'); xlabel('Q'), ylabel('R1 (ohms)');**

**grid on;**

**subplot(2,1,2);**

**plot(Q,R2,'LineWidth',2);**

**title('R2 vs Q'); xlabel('Q'), ylabel('R2 (ohms)');**

**grid on;**

**R1min = min(R1); R1max = max(R1);**

**R2min = min(R2); R2max = max(R2);**

**R1min**

**R1max**

**R2min**

**R2max**

**C = 10e-9; n = 100; Q = 8;**

**f = @(wn) sallenKeyComponents(C,n,Q,wn);**

**wn\_min = 1e3; wn\_max = 30e3;**

**wn = linspace(wn\_min,wn\_max,1001);**

**Y = zeros(1001,4);**

**R1=0;R2=0;C1=0;C2=0;**

**for i=1:1001**

**[R1,R2,C1,C2] = f(wn(i));**

**Y(i,:) = [R1,R2,C1,C2];**

**end**

**R1 = Y(:,1); R2 = Y(:,2);**

**subplot(2,1,1);**

**plot(wn,R1,'LineWidth',2);**

**title('R1 vs wn'); xlabel('Q'), ylabel('R1 (ohms)');**

**grid on;**

**subplot(2,1,2);**

**plot(wn,R2,'LineWidth',2);**

**title('R2 vs wn'); xlabel('Q'), ylabel('R2 (ohms)');**

**grid on;**

**R1min = min(R1); R1max = max(R1);**

**R2min = min(R2); R2max = max(R2);**

**R1min**

**R1max**

**R2min**

**R2max**

### 04-03-2021 SAT

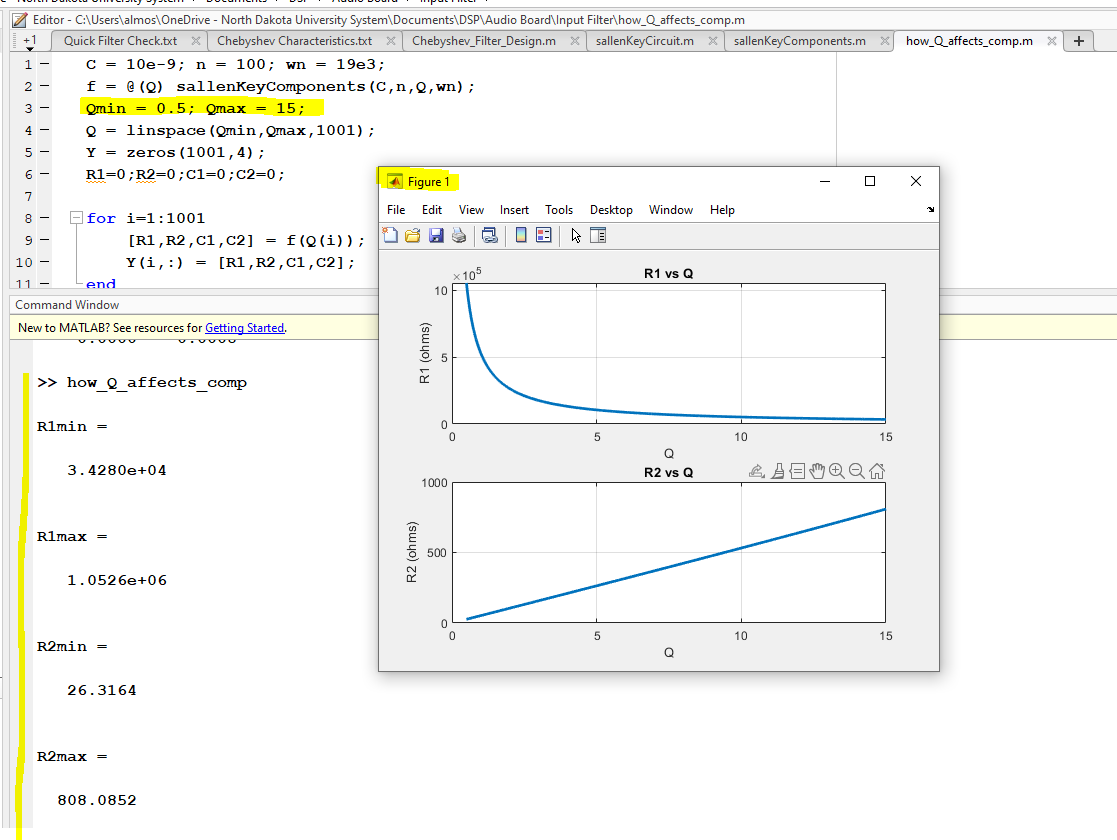
Current Stage: Stage A step 2

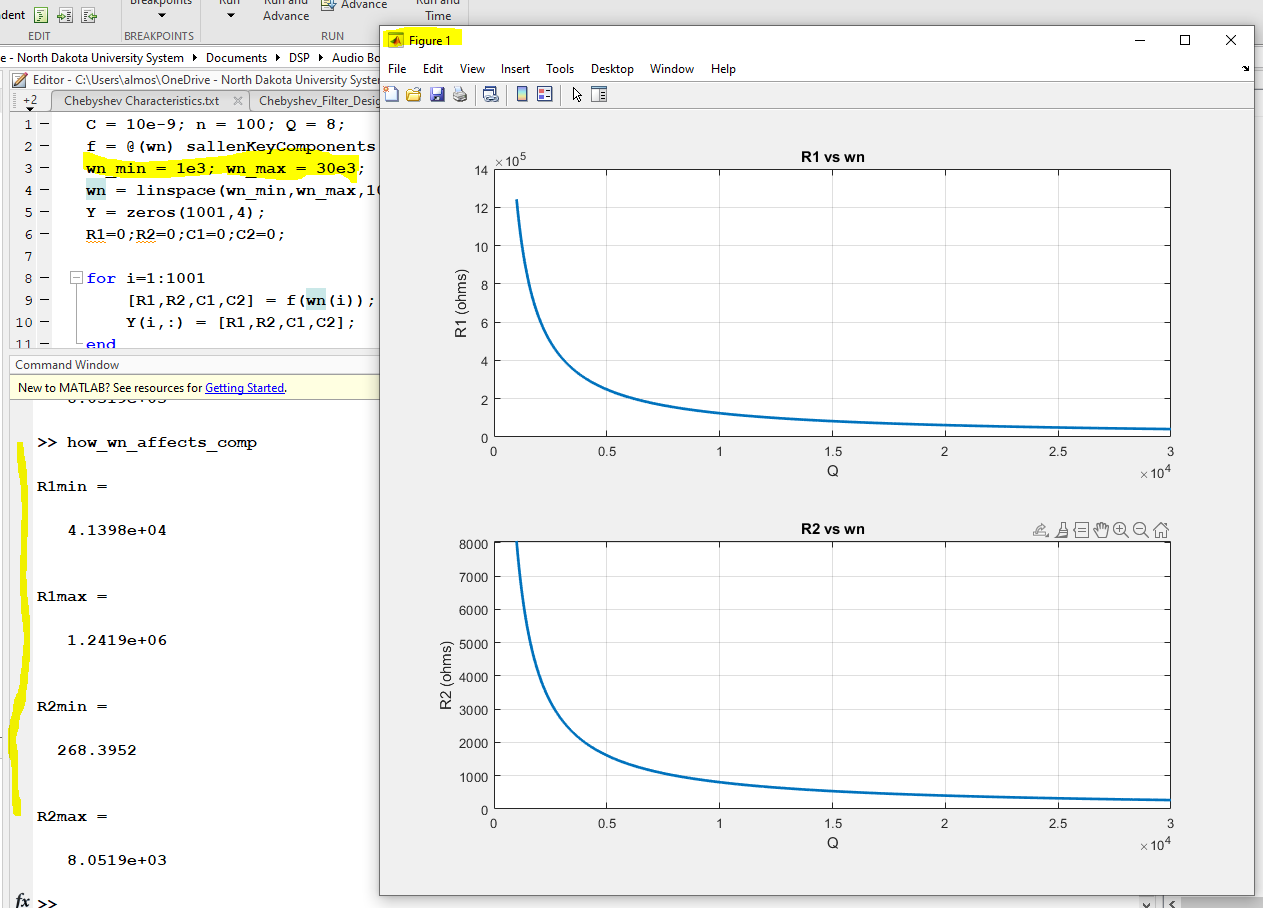
Start Time: 4:15 PM

End Time: 12:44 AM (-3 hours for other stuff)

(5h29m) (14h29m)

* + - * Continued working on MATLAB script to turn filter design into Sallen Key stages. Learned a few things.





**function [sorted\_list] = sort\_complex\_list(list)**

**% This sort assumes list has conjugate pairs**

**% Inputs**

**% list --> This is the complex list of conjugate pairs to be sorted**

**% Outputs**

**% sorted\_list --> A N/2 rows by 2 columns list with conjugate pairs**

**% in a row, sorted from largest to smallest in**

**% magnitude**

**% First get rid of conjugate pairs cuz that makes this sorting more**

**% complicated**

**list = reshape(list,1,[]);**

**N = length(list);**

**i=1;**

**while i<=N**

**j=1;**

**% fd1 = list(i);**

**while j<=N**

**% fd2 = list(j);**

**if conj\_equality(list(i),list(j)) && abs(imag(list(j)))>= 1e-4**

**list = remove\_index(j,list);**

**N = N-1;**

**end**

**j=j+1;**

**end**

**i=i+1;**

**end**

**% Now sort in descending order of magnitude**

**swap = 1;**

**while swap==1**

**swap = 0;**

**for i=1:length(list)-1**

**if abs(list(i)) < abs(list(i+1))**

**temp = list(i+1);**

**list(i+1) = list(i);**

**list(i) = temp;**

**swap = 1;**

**elseif list(i) == -list(i+1)**

**end**

**end**

**end**

**% Now we make a (something)-by-2 matrix with each row being a complex**

**% number and its conjugate**

**sorted\_list = zeros(length(list),2);**

**for i=1:length(list)**

**sorted\_list(i,1) = list(i);**

**sorted\_list(i,2) = conj(list(i));**

**end**

**end**

* + - * I made a MATLAB function file for computing Q and wn given the poles of a 2nd order stage:

**function [Q, wn, alpha, zeta] = sallenKeyCircuit2(poles)**

**% Inputs**

**% poles --> Poles of 2nd order filter stage --> Assumed to be conjugate**

**% pairs!!**

**% Outputs**

**% Q --> Quality factor**

**% zeta --> Filter natural frequency**

**% alpha --> Quality factor**

**% wn --> Damping factor**

**th = pi - angle(poles(1));**

**zeta = cos(th);**

**Q = 1/(2\*zeta);**

**wn = abs(poles(1));**

**alpha = wn/(2\*Q);**

**end**

* + - * Using this function, I am now able to go from designed poles to filter stage component values by adding in the following into the Chebyshev\_Filter\_Design.m file, **(woo!)**

**%% Compute components for filter implemented as a cascade of Sallen-key stages**

**% !!This assumes even order and no poles on real-axis!!**

**% Use sallenKeyCircuit2.m function to obtain Q and wn associated with each**

**% stage**

**sorted\_poles = sort\_complex\_list(pk);**

**NN = length(sorted\_poles);**

**Q = zeros(1,NN); wn = zeros(1,NN);**

**for i=1:NN**

**[Q(i), wn(i), aa, zz] = sallenKeyCircuit2(sorted\_poles(i,:));**

**end**

**% Use sallenKeyComponents to get R1,R2,C1,C2 of each stage given Q and wn**

**C = 10e-9; n = 100;**

**R1 = zeros(1,NN); R2 = zeros(1,NN);**

**C1 = zeros(1,NN); C2 = zeros(1,NN);**

**for i=1:NN**

**[R1(i), R2(i), C1(i), C2(i)] = sallenKeyComponents(C,n,Q(i),wn(i));**

**end**

And within the script I have it print out to a text file the most relevant info, and here is the one with the current design:

**CHEBYSHEV FILTER**

**fp: 3042 Hz fs: 4000 Hz**

**delta\_p: 0.90 delta\_s: 0.01**

**ap: 0.92 dB as: 40.00 dB**

**epsilon: 0.484322**

**Filter Order: 8**

**Poles (Hz):**

**-109.828560 +- 3034.136883**

**-312.765276 +- 2572.217047**

**-468.086315 +- 1718.700483**

**-552.145455 +- 603.527350**

**Stages:**

**Stage 1: Q: 13.822 wn: 3036.12 Hz R1: 37186.07 ohms R2: 738.96 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 2: Q: 4.142 wn: 2591.16 Hz R1: 148024.02 ohms R2: 254.87 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 3: Q: 1.903 wn: 1781.30 Hz R1: 469400.72 ohms R2: 170.07 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 4: Q: 0.741 wn: 817.99 Hz R1: 2626530.24 ohms R2: 144.13 ohms C1: 1000.00 nF C2: 0.10 nF**

* + - * So now for actual component values to use, I think I’ll for the E12 series of values because those are most likely to be found in the lab. However, in the lab there are additional values. Here’s my choice for now:

E12 series decade: 10 12 15 17 22 27 33 39 47 56 68 82 (I did see in the lab 75 is also available)

IN THE LAB: 10 12 15 18 20 22 27 30 33 39 47 51 56 62 68 75 82 91

Stages:

Stage 1: Q: 13.822 wn: 3036.12 Hz R1: 37186.07 --> **39k ohms** R2: 738.96 --> **750 ohms** C1: 1000.00 nF C2: 0.10 nF

Stage 2: Q: 4.142 wn: 2591.16 Hz R1: 148024.02 --> **150kohms** R2: 254.87 --> **270 ohms** C1: 1000.00 nF C2: 0.10 nF

Stage 3: Q: 1.903 wn: 1781.30 Hz R1: 469400.72 --> **470kohms** R2: 170.07 --> **180 ohms** C1: 1000.00 nF C2: 0.10 nF

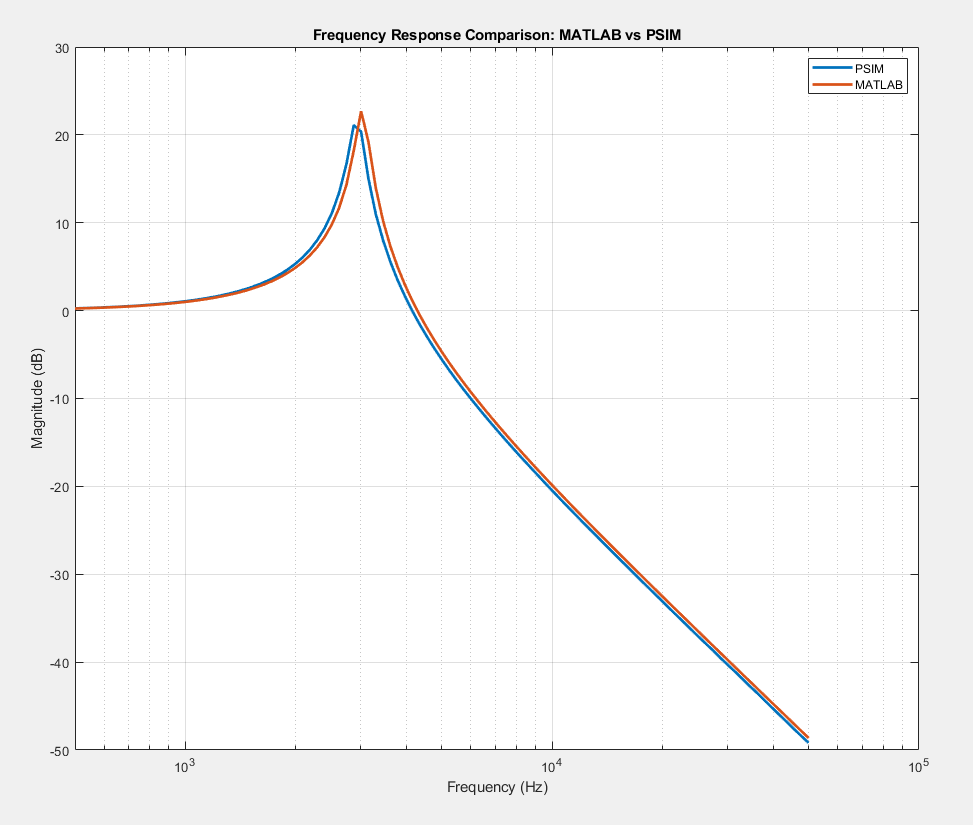
Stage 4: Q: 0.741 wn: 817.99 Hz R1: 2626530.24 --> **1 + 1 + 0.68 Mohms** R2: 144.13 --> **150 ohms** C1: 1000.00 nF C2: 0.10 nF

* + - * Based on chosen values, here are all the simulation results and comparisons:

Stages: 1

**Peak of Original Design: 13.62 23 dB**

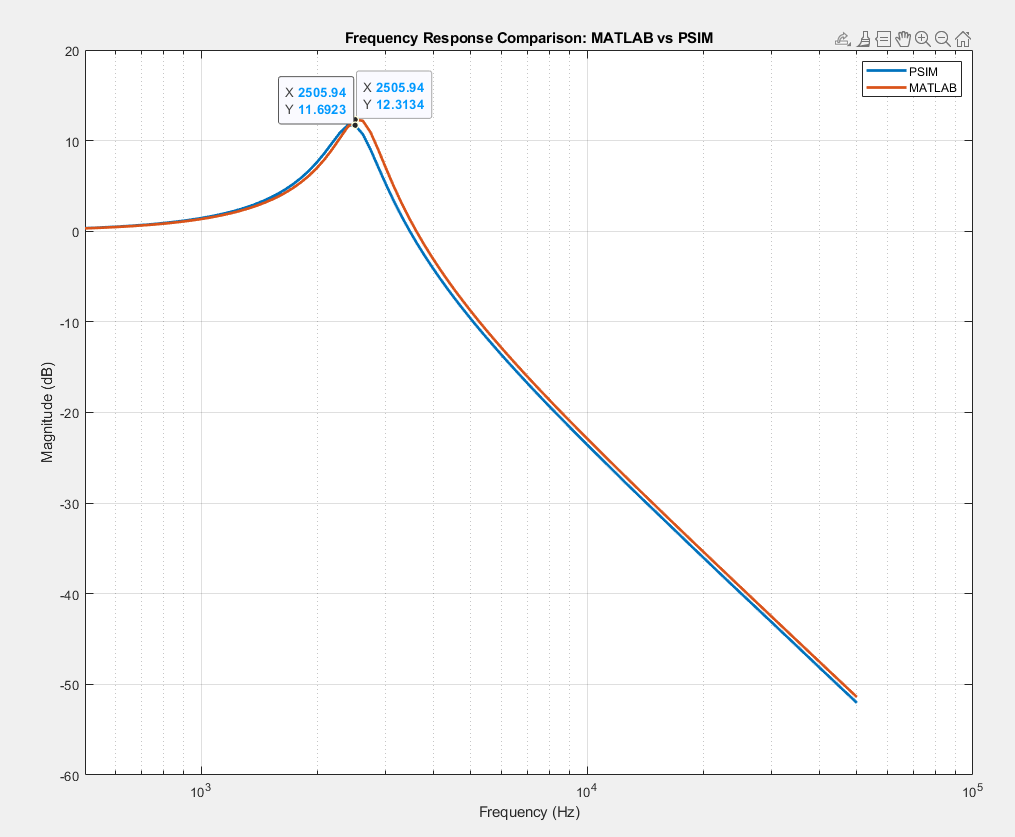
**Peak of PSIM simulation with standardized values: 11.35 21 dB**



Stages: 2

**Peak of Original Design: 4.13 12 dB**

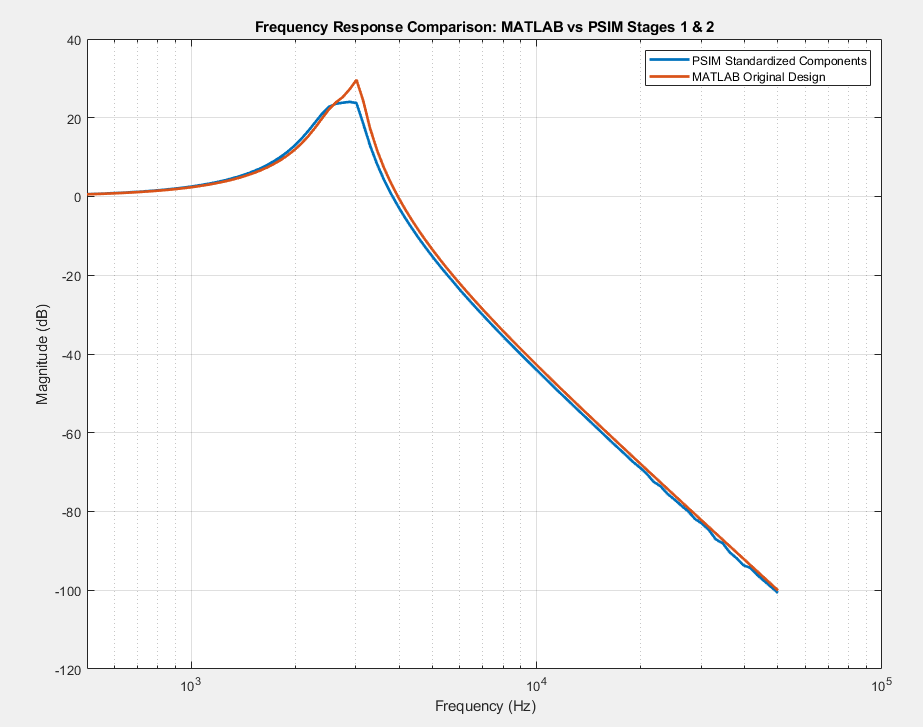
**Peak of PSIM simulation with standardized values: 3.84 12 dB**



Stages: 1 and 2

**Peak of Original Design: 30.26 30 dB**

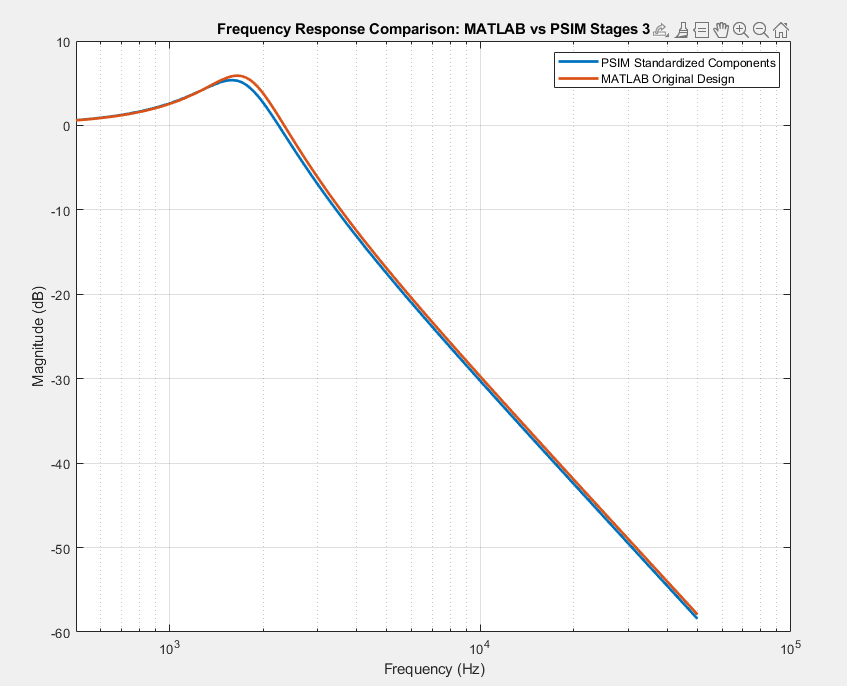
**Peak of PSIM simulation with standardized values: 15.92 24 dB**



Stages: 3

**Peak of Original Design: 1.97 6 dB**

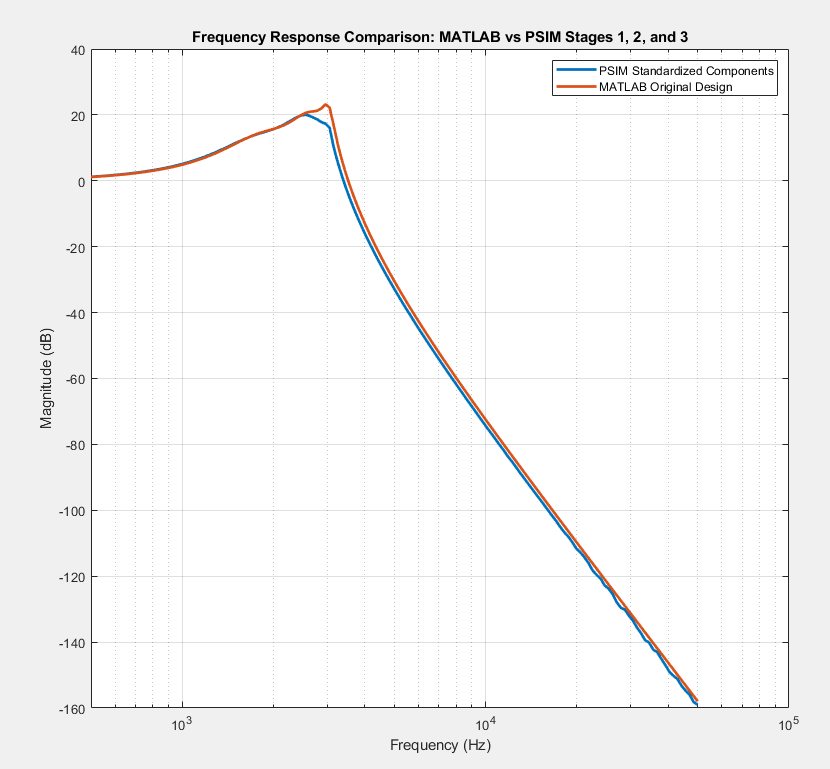
**Peak of PSIM simulation with standardized values: 1.85 5 dB**



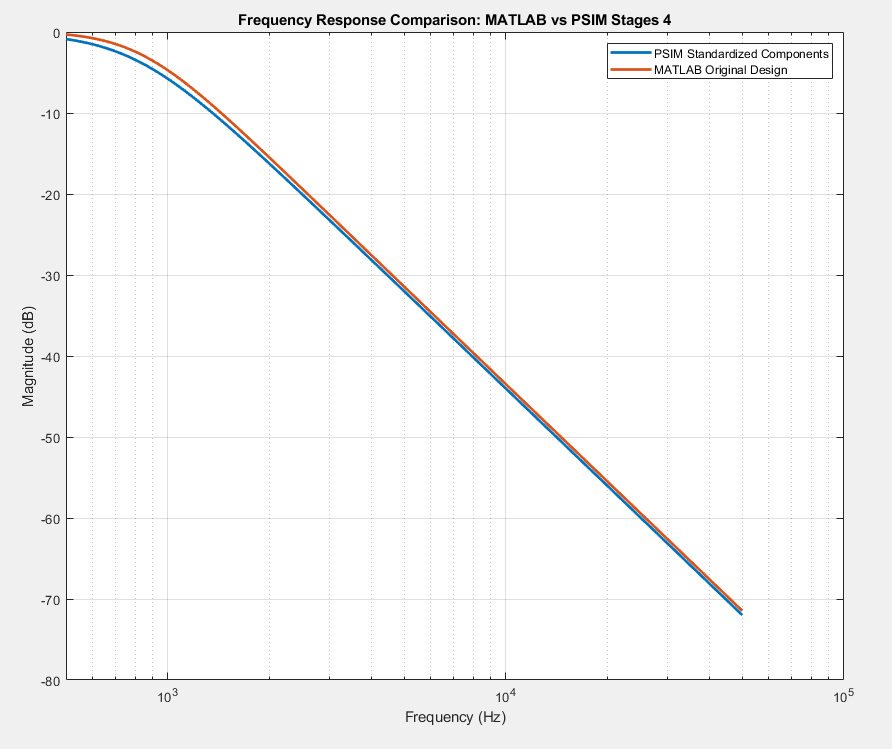
Stages: 1, 2, and 3

**Peak of Original Design: 14.49 23 dB**

**Peak of PSIM simulation with standardized values: 10.07 20 dB**



Stages: 4



**We can see here how the overall filter will maintain <1 gain over its spectrum because this last stage attenuates all the previous stages!**

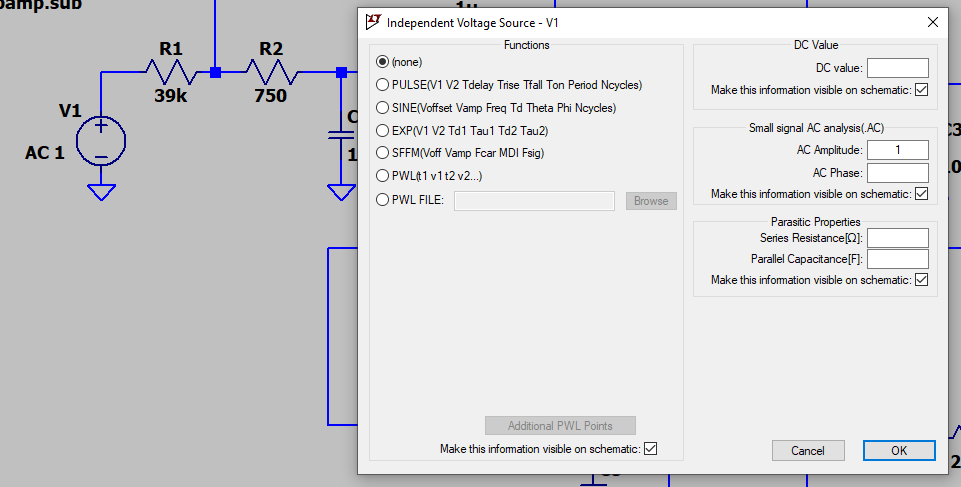
*Now all together!!*

So here PSIM 64-bit Demo Version 2021a unfortunately does not allow for a simulation that includes more than 15 RLC components, and for doing four stages of a Sallen-Key circuit, I have 16 RLC components… So I’ll try LTspice here.

LTspice is certainly a different feel, and took a sec to get some details worked out:

For an op amp, I used the component opamp, but in order to use it, an .include opamp.sub SPICE directive was needed!

To run an AC sweep, the voltage source to be used should be…



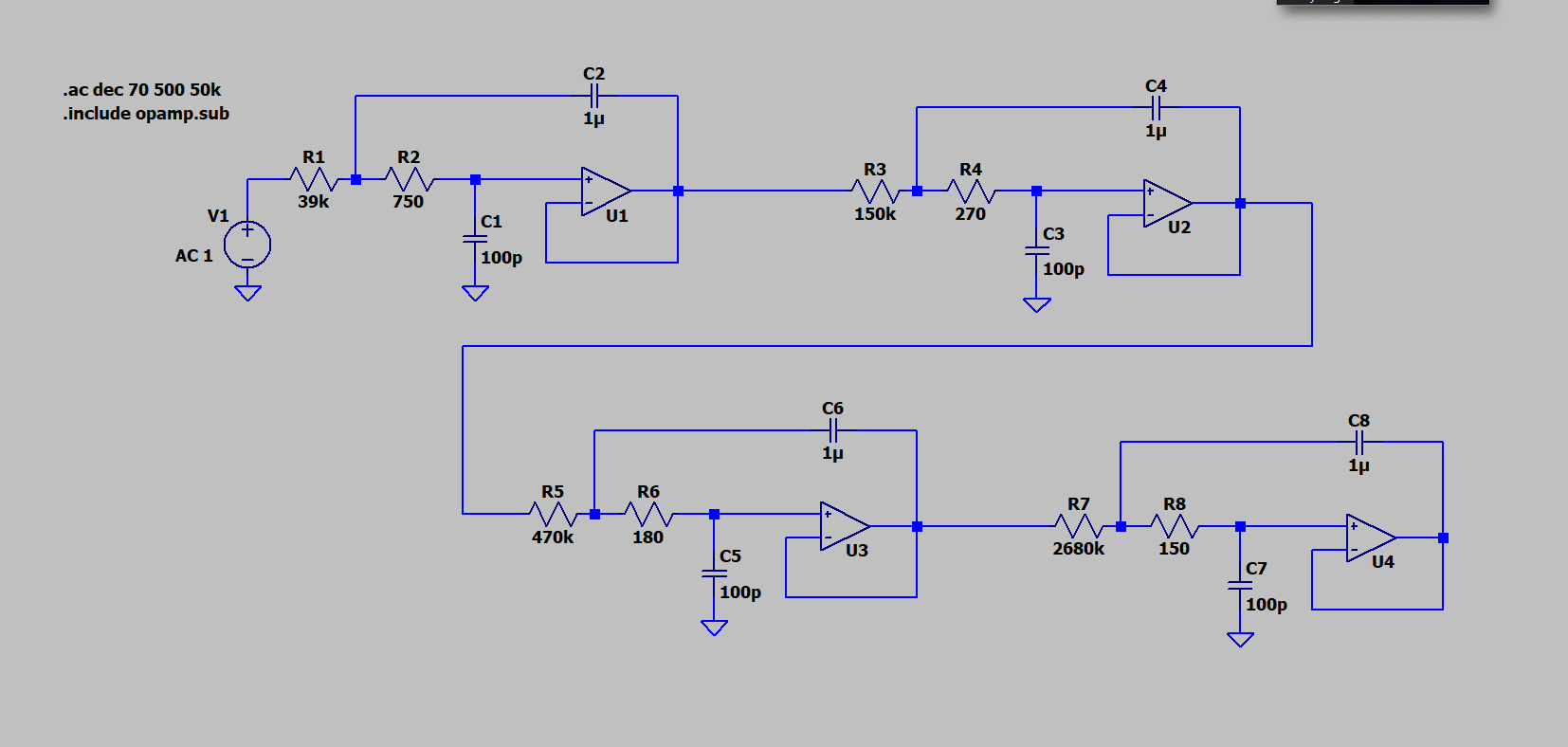
Finally, to export the data from the AC sweep, within LTspice, unfortunately it lays out the magnitude and phase like this: **(mag dB, phase°),** which **makes it a little more difficult to read into MATLAB**. **So, the solution is to firstly delete the first row in the text file that has the names of the columns, and then use the following code:**

**fid = fopen('8th\_Order\_LP\_Cheyshev.txt');**

**Dc = textscan(fid, '%f(%fdB,%f°)', 'CollectOutput',1);**

**A = cell2mat(Dc);**

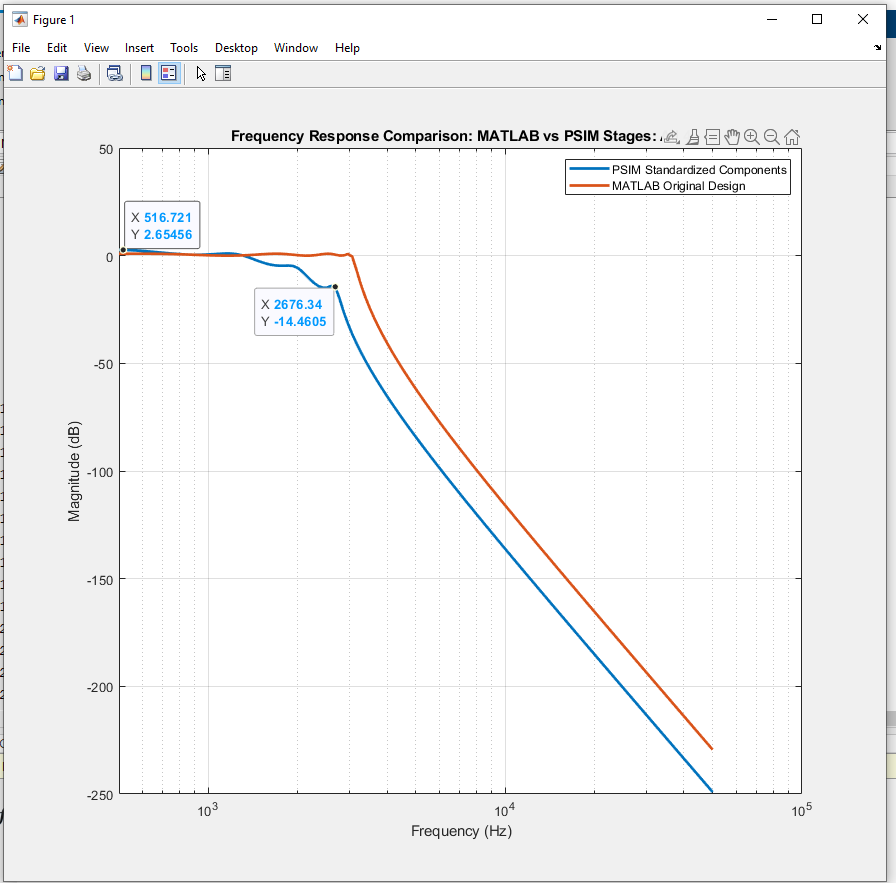
Here was my layout in the end:



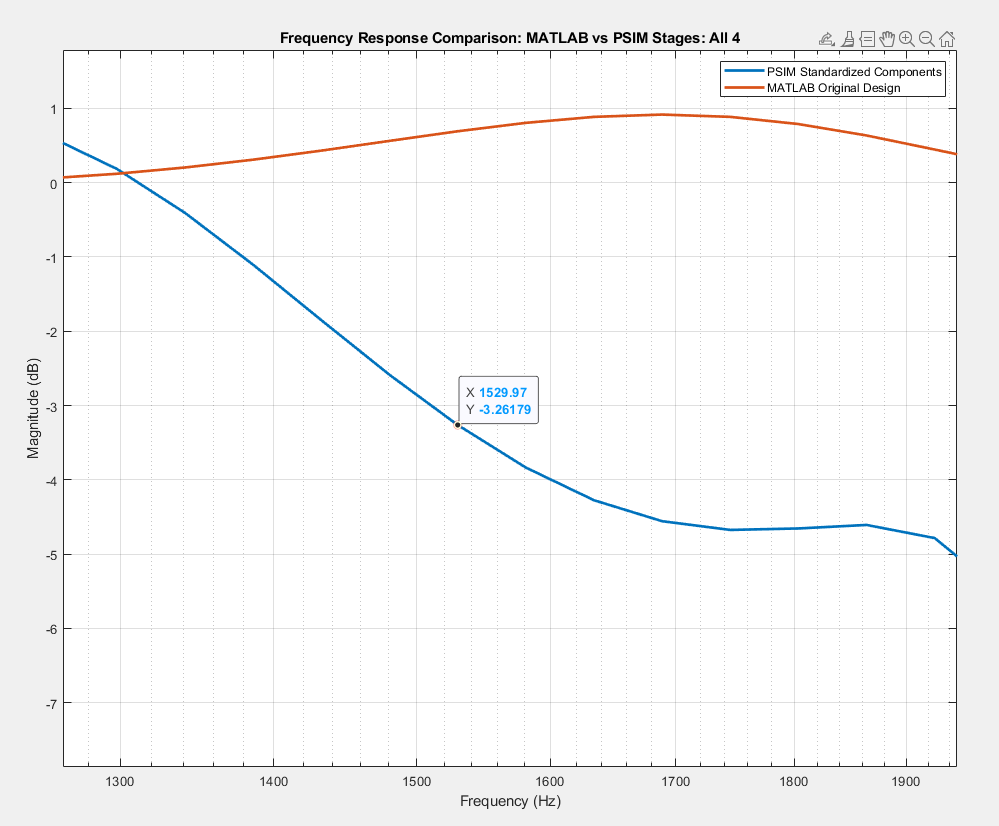
REALLY glad this worked tho! It was actually quicker than PSIM and without the component limit! :D

**Peak of Original Design: 1.11 1 dB**

**Peak of PSIM simulation with standardized values: 1.36 3 dB**



Ok, so we can see that because each stage’s magnitude response was a little lower than the original design’s, we find that they add together to make the resulting curve lower-offset than the original design, and because of this the cutoff frequency is no longer the ~3kHz mark but more like 1.5kHz…



This is kinda no-bueno… Even though the individual stages we fairly close to each other, the accumulative effect is certainly showing itself now…

**So I’ll redo these steps but for a ~6kHz cutoff frequency and see what happens. Hopefully, the automation I’ve put into this development process will make this second run much faster!**

Doing a quick re-design… Let’s see!

**CHEBYSHEV FILTER**

**fp: 6084 Hz fs: 8000 Hz**

**delta\_p: 0.90 delta\_s: 0.01**

**ap: 0.92 dB as: 40.00 dB**

**epsilon: 0.484322**

**Filter Order: 8**

**Poles (Hz):**

**-219.657119 +- 6068.273767**

**-625.530552 +- 5144.434095**

**-936.172630 +- 3437.400966**

**-1104.290910 +- 1207.054701**

**Stages:**

**Stage 1: Q: 13.822 wn: 6072.25 Hz R1: 18593.03 ohms R2: 369.48 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 2: Q: 4.142 wn: 5182.32 Hz R1: 74012.01 ohms R2: 127.44 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 3: Q: 1.903 wn: 3562.60 Hz R1: 234700.36 ohms R2: 85.03 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 4: Q: 0.741 wn: 1635.98 Hz R1: 1313265.12 ohms R2: 72.07 ohms C1: 1000.00 nF C2: 0.10 nF**

Surprisingly, not much different!! Based on the shown component values, I’ll choose the components as follows:

**Stages:**

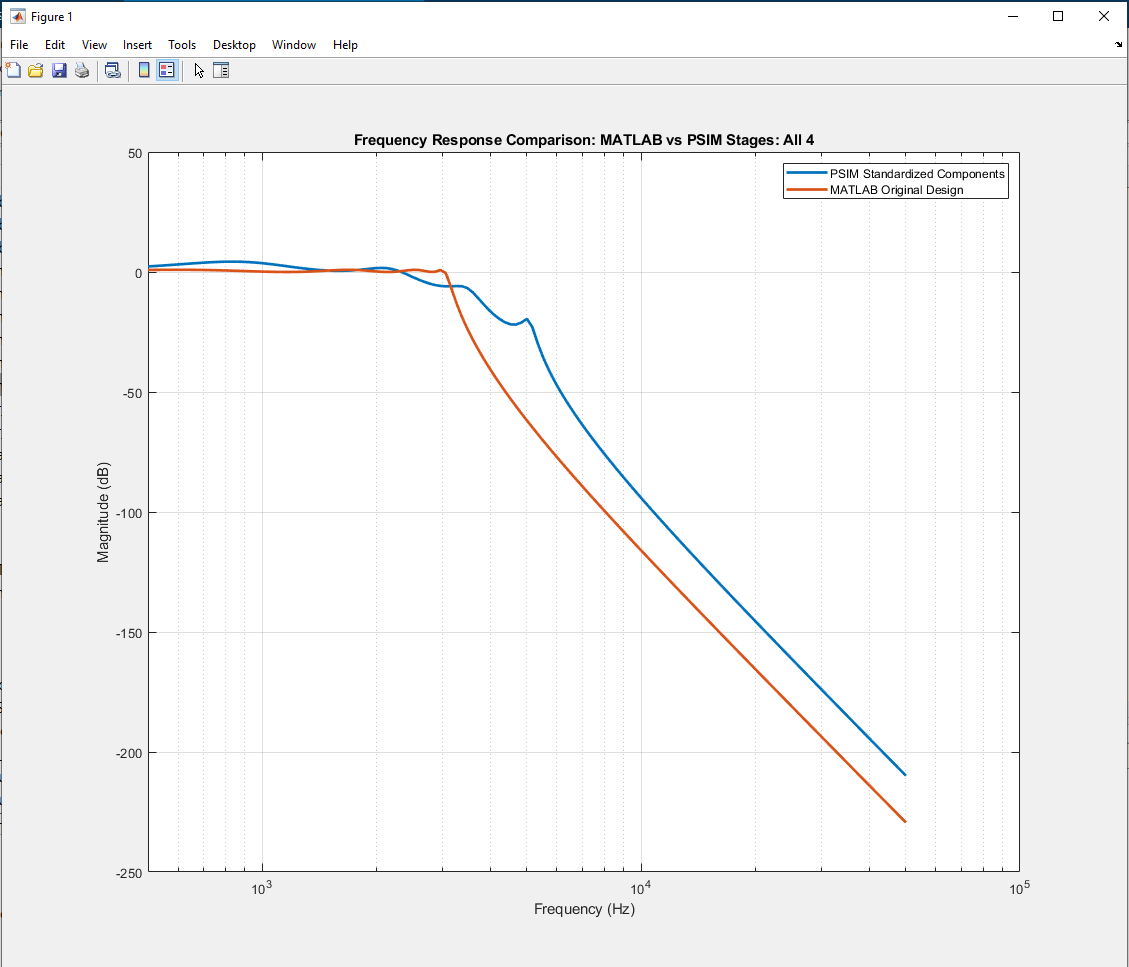
**Stage 1: Q: 13.822 wn: 6072.25 Hz R1: 18593.03 --> 18k ohms R2: 369.48 --> 390 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 2: Q: 4.142 wn: 5182.32 Hz R1: 74012.01 --> 75k ohms R2: 127.44 --> 120 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 3: Q: 1.903 wn: 3562.60 Hz R1: 234700.36 --> 220k ohms R2: 85.03 --> 82 ohms C1: 1000.00 nF C2: 0.10 nF**

**Stage 4: Q: 0.741 wn: 1635.98 Hz R1: 1313265.12 --> 1M + 300k ohms R2: 72.07 --> 75 ohms C1: 1000.00 nF C2: 0.10 nF**

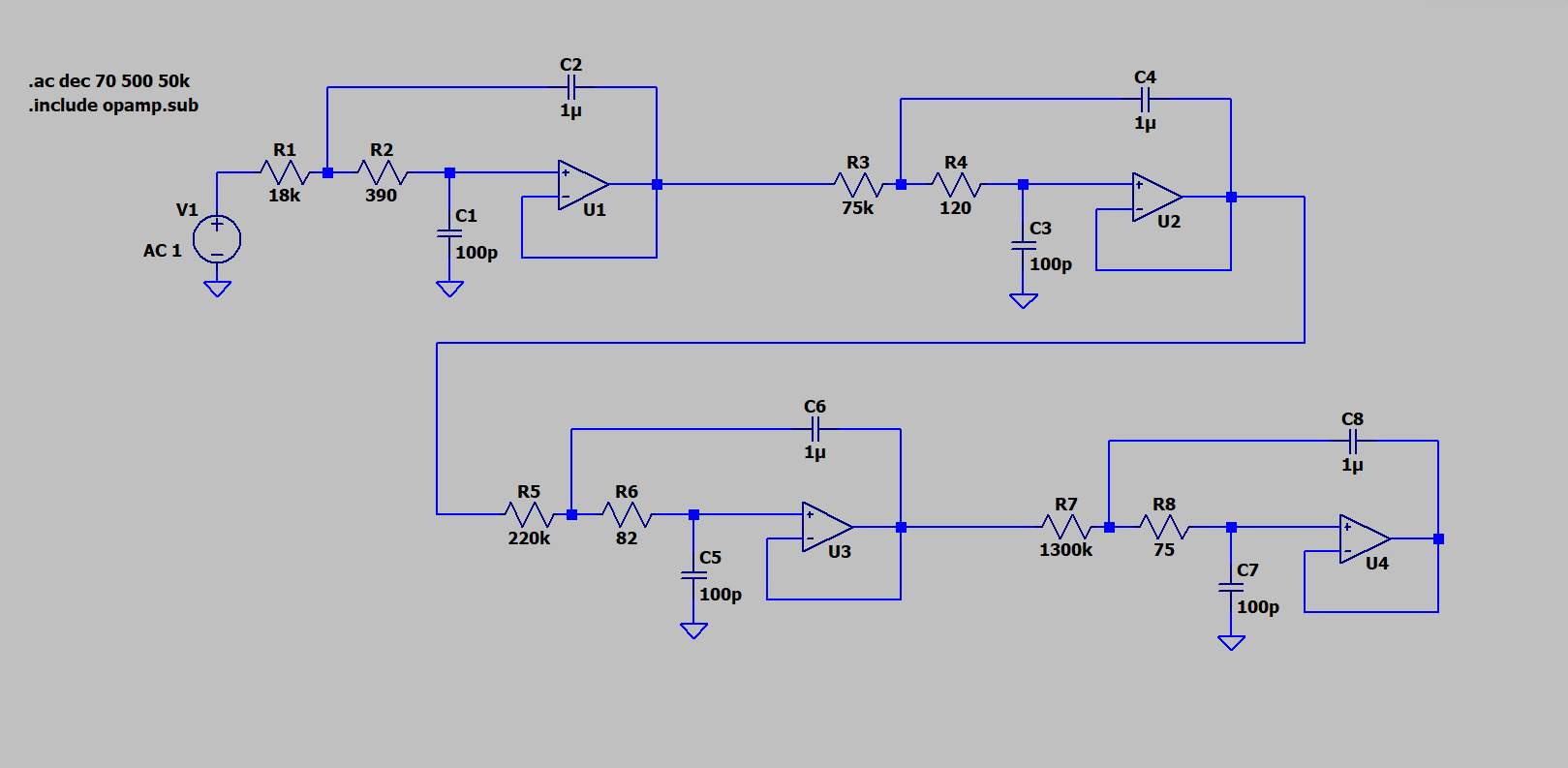
Putting this into LTspice and then into MATLAB to compare…



**Peak of Original Design: 1.11 1 dB**

**Peak of PSIM simulation with standardized values: 1.64 4 dB**

So, alright! We’re doing a LOT better in the passband, with the cutoff still short (~2.5kHz) but closer to the designed cutoff. However, in the stopband, we are little offset (~+20dB), but I think I can live with that. So alright**, version 2 of design**:



### 04-04-2021 SUN

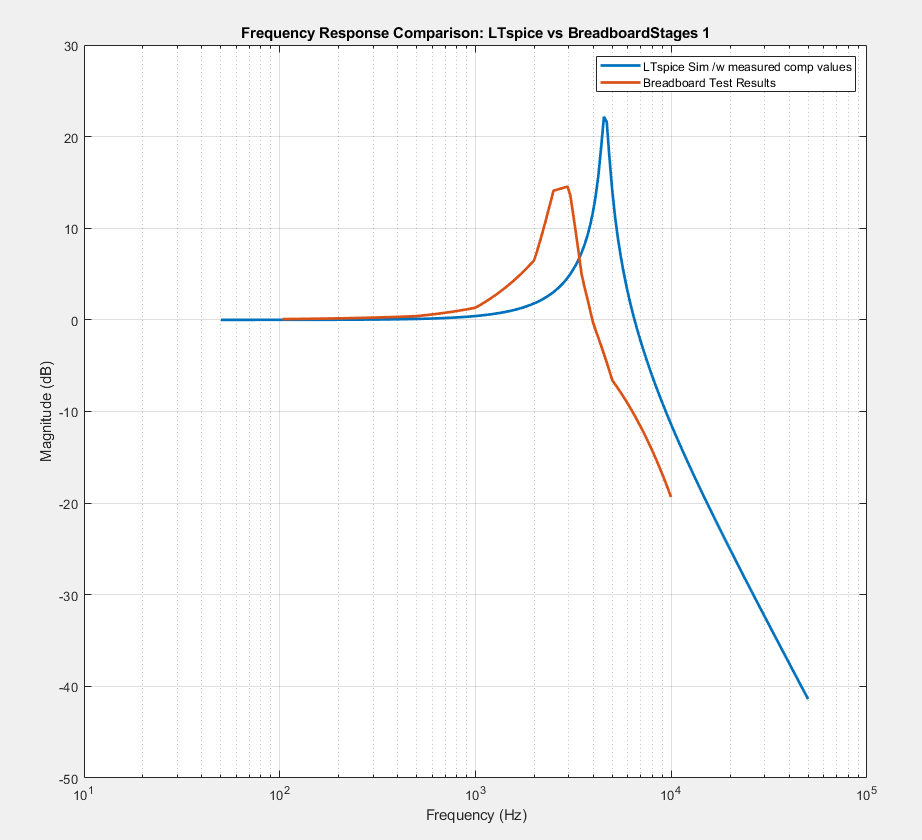
Current Stage: Stage B1

Time Start: 10:00 AM

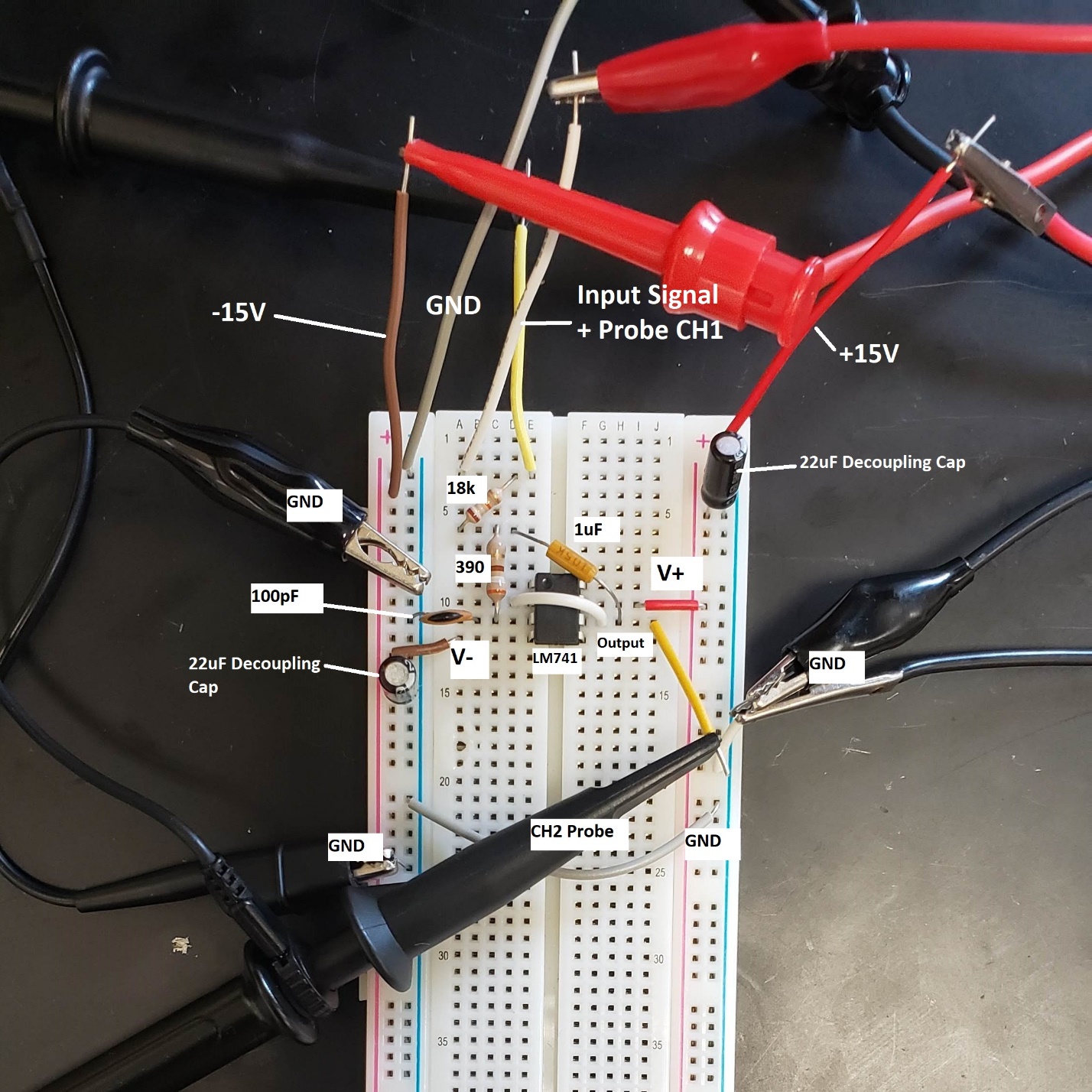
Time End: 5:00 PM

(7h) (21h29m)

* + - * Off to the lab and the breadboard!
      * Just for first stage, got components, measured their true values with the desktop multimeter, and wrote that down, along with other info.
      * Made a quick op-amp test rig (i.e., inverting amp circuit for x2 amp) and testing two of the op amps.
      * Realized that there was something wrong with the circuit on the breadboard because the gain from 100Hz to 50kHz was pretty much constant at 1 and there was no phase change at all…
        1. After switching each component on the board one-by-one, trying another Sallen-key circuit, and discovering that two of the lab power supplies had a tough time with setting up a series connection (which I used to get the +-15V), it turned out to be the breadboard somehow. I basically stripped off all the components and put the same circuit onto another breadboard and now it worked just fine. UGGHHH.
      * After testing from 100Hz to 10kHz, found that the results **did not quite match up with the expected results from LTspice (despite accounting for accurate component values)**, as shown below:



This was my breadboard setup:



I actually went ahead and inquired as to why there was this much discrepancy on Reddit: [link](https://www.reddit.com/r/ElectricalEngineering/comments/mk4qrf/breadboard_circuit_vs_simulated_circuit_3/). It seems the suggestions so far hint at improving the accuracy of the simulation by using a more detailed op amp model rather than the ideal opamp component I used from the LTspice library. 🡪 **In any case, I may have to do a ver3 redesign that pushes the cutoff frequency even higher to account for the potential left-shift of the breadboard frequency response, so that at least on the breadboard, I more closely match the standardized-version of my original design…**

I did also write a separate MATLAB script for more easily comparing the LTspice simulation plot to the experimental results plot, and this was used to generate the plot shown above.

* + - * Aside from this, I also found from the Wikipedia page on the Sallen-Key topology about the input impedance of this circuit and wrote up a sallenKeyInputImp.m file for plotting the input impedance of a given circuit. **This will be important to know as I initially planned for the flowmeter sensors to have enough drive to push through this filter stage, but given my first stage seems to have a minimum input impedance of around 1k,** **I may want to go for an input buffer...**
      * **It is important for me to keep track of three types of the same circuit:**
        1. **Type I** - Original design from the ChebyshevFilterDesign.m script
        2. **Type II** - Design that standardizes the component values from the original design
        3. **Type III** - Design that is for the breadboard that is meant to as closely as possible match Type II.
      * **So, for tomorrow, I will probably**
        1. **Redesign the circuit for its Type III version to have a nominally higher cutoff frequency, while comparing the results to the Type II version**
        2. **Improve the LTspice model I have for the op amp**

### 04-10-2021 MON

Current Stage: Stage B step 1d

Start Time:

End Time:

() ()