An Efficient Implementation of the Patterson-Holdsworth Auditory Filter Bank

Malcolm Slaney
Apple Computer Technical Report #35
Perception Group—Advanced Technology Group

1.0 Introduction

This report describes an implementation of a cochlear model proposed by Roy Patterson [Patterson1992]. Like a previous report [Slaney1988], this document is an electronic notebook written using a software package called *Mathematica*™ [Wolfram 1988].

This report describes the filter bank and its implementation. The filter bank is designed as a set of parallel bandpass filters, each tuned to a different frequency. This report extends previous work by deriving an even more efficient implementation of the Gammatone filter bank, and by showing the MATLAB™ code to design and implement an ERB filter bank based on Gammatone filters.

2.0 Ear Filters

Patterson's cochlear model is based on an array of independent bandpass filters. The filters are tonotopically organized from high frequencies at the base of the cochlea, to low frequencies at the apex. In Patterson's model the bandwidth of each cochlear filter is described by an Equivalent Rectangular Bandwidth (ERB) [Patterson1992]. This section of the report describes how cochlear channels are spaced so that each filter overlaps its neighbors by the same amount.

2.1 Equivalent Rectangular Bandwidth (ERB)

The ERB is a psychoacoustic measure of the width of the auditory filter at each point along the cochlea. Psychoacoustic experiments often measure something known as the Critical Band but we will use the terms interchangeably. A critical band or ERB filter models the signal that is present within a single auditory nerve cell or channel. Glasberg and Moore [Glasberg1990] recommend the following equation for the ERB.

```
ERB[f_] := 24.7(4.37 f / 1000 + 1)

ERB[1000]

132.639

ERB[3000]

348.517
```

This can be rewritten as

$$\mathrm{ERB}[f\_, \mathrm{EarQ}\_, \mathrm{minBW}\_] := \frac{f}{\mathrm{EarQ}} + \mathrm{minBW};$$

where *EarQ* is the asymptotic filter quality at large frequencies and *minBW* is the minimum bandwidth for low frequencies channels. In between there is a smooth transition. This is similar to the bandwidth defined in Lyon's model which is given by

$$\frac{\mathrm{Sqrt}[f^2 + \mathrm{EarBreakFreq}^2]}{\mathrm{EarQ}}$$

A general form for the bandwidth of a cochlear channel as a function of center frequency is given by

```
ERB[f_, EarQ_, minBW_, order_] :=
              ((f/EarQ)^order + (minBW)^order)^(1/order);
```

with Glasberg and Moore recommending the following parameters

```
GlasbergBandwidthParms={EarQ->1000/(24.7*4.37),
                  minBW->24.7*1,
                  order->1}

{EarQ -> 9.26449, minBW -> 24.7, order -> 1}
```

Lyon [Slaney1988] recommends a nominal bandwidth given by

```
LyonBandwidthParms={minBW->EarBreakF/8,
                    EarQ->8,
                    order->2}/.
                        EarBreakF->1000
```

```
{minBW -> 125, EarQ -> 8, order -> 2}
```

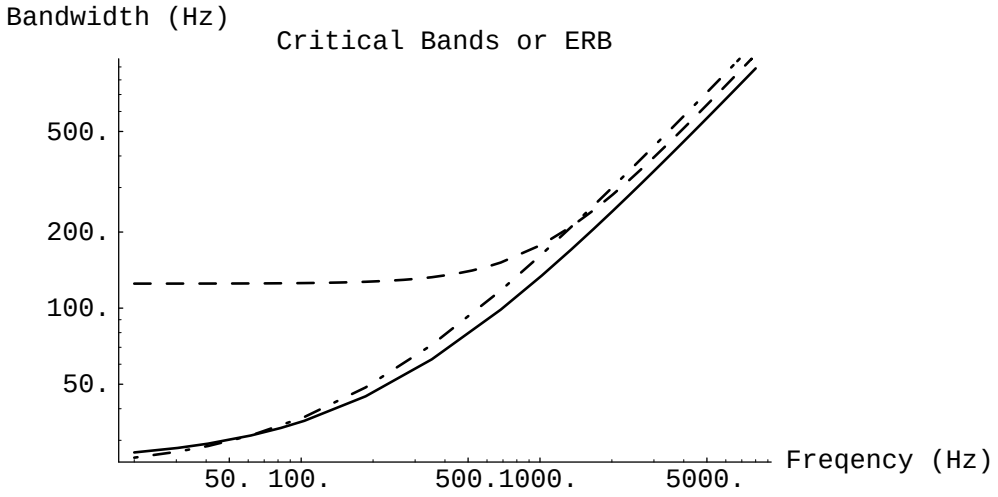
and Greenwood [Greenwood1990] recommending

```
GreenwoodBandwidthParms={EarQ->35/(Log[10]*a),
                         minBW->a*A*k*Log[10]/35,
                         order->1}/.
                             a->2.1/.
                             A->165.4/.
                             k->1//N
```

```
{EarQ -> 7.23824, minBW -> 22.8509, order -> 1.}
```

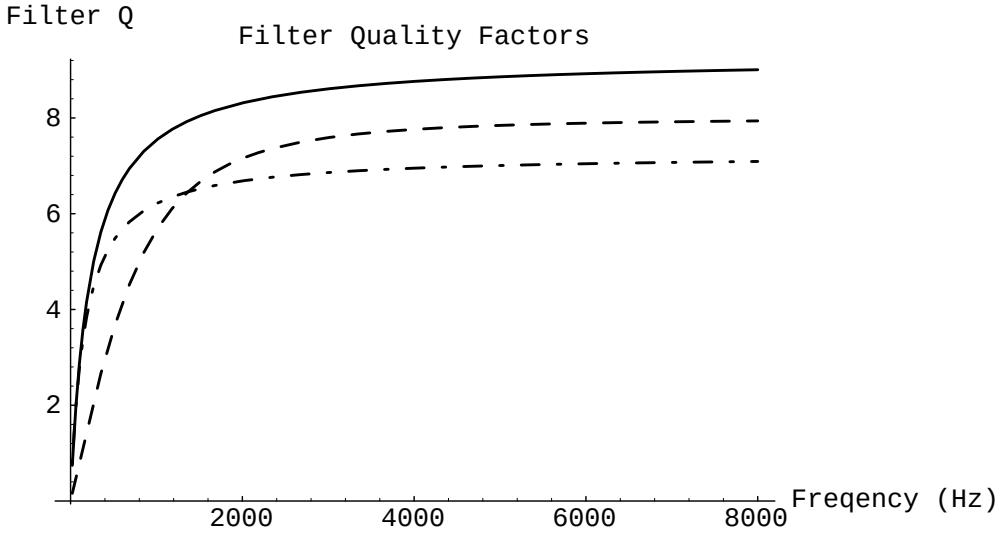
These three bandwidth functions are plotted below. The solid curve is the Critical Band function recommended by Glasberg and Moore, the dashed line is the one by Lyon, and the dash-dot line is from Greenwood.

```
<<Graphics`Graphics`
Show[{LogLogPlot[ERB[f,EarQ,minBW,order]/.GlasbergBandwidthParms,
            {f,20,8000},DisplayFunction->Identity,
            AxesLabel->{"Freqency (Hz)","Bandwidth (Hz)"},
            PlotLabel->"Critical Bands or ERB"],
     LogLogPlot[ERB[f,EarQ,minBW,order]/.LyonBandwidthParms,
            {f,20,8000},DisplayFunction->Identity,
            PlotStyle->Dashing[{0.025,0.025}]],
     LogLogPlot[ERB[f,EarQ,minBW,order]/.GreenwoodBandwidthParms,
            {f,20,8000},DisplayFunction->Identity,
            PlotStyle->Dashing[{0.005,0.025,0.025,0.025}]]},
     DisplayFunction->$DisplayFunction];
```



The equivalent filter quality factor, Q, can be determined by dividing center frequency, f, by the ERB. In this plot, the highest curve (sharper filters) corresponds to Glasberg's suggested parameters. The solid curve is the Critical Bank function recommended by Glasberg and Moore, the dashed line is the one by Lyon, and the dash-dot line is from Greenwood

```
Show[{Plot[f/ERB[f,EarQ,minBW,order]/.GlasbergBandwidthParms,
            {f,20,8000},DisplayFunction->Identity,
            AxesLabel->{"Freqency (Hz)","Filter Q"},PlotRange->All
            PlotLabel->"Filter Quality Factors"],
     Plot[f/ERB[f,EarQ,minBW,order]/.LyonBandwidthParms,
            {f,20,8000},DisplayFunction->Identity,
            PlotStyle->Dashing[{0.025,0.025}]],
     Plot[f/ERB[f,EarQ,minBW,order]/.GreenwoodBandwidthParms,
            {f,20,8000},DisplayFunction->Identity,
            PlotStyle->Dashing[{0.005,0.025,0.025,0.025}]]},
     DisplayFunction->$DisplayFunction];
```



In all three cases the Q is flat, except at low frequencies where the auditory filters have a smaller Q and a wider relative bandwidth. For the rest of this report we will use Glasberg's parameters.

■ 2.2 Channel Spacing

Each filter in Paterson's model is one ERB wide, but the frequency spacing between channels is not specified. In the cochlea there are many thousands of hair cells but computer models can only approximate this density of channels. Two ways to allocate the channels will be described here. In the simpler case, each cochlear channel is spaced a given fraction of an ERB from the previous one. Alternatively, the highest and lowest frequencies can be specified along with the desired number of channels. The equation for the simple case can then be solved to find the proper ERB spacing for each channel.

In our implementation of Lyon's Cochlear Model [Slaney1988] we used the parameter *stepfactor* to indicate the amount of overlap (usually as a fraction less than one.) We will use the same

convention here. *Stepfactors* close to zero indicate filters nearly completely overlap, a *stepfactor* of 0.5 means that each frequency in the input signal is "sampled" by two cochlear channels, and a *stepfactor* of 1 means that there is almost no overlap between channels.

The following expression represents the mapping between center frequency and cochlear position index. We integrate from the Nyquist frequency (*fn*) down to any arbitrary frequency (*cf*) to model the propagation of energy from base (high frequencies) to apex (low frequencies).The resulting equation gives us the channel number that is centered on any arbitrary frequency.

```
index = Integrate[-1/ERB[f, EarQ, minBW, 1]/
                    stepfactor,{f,fn,cf}]
```

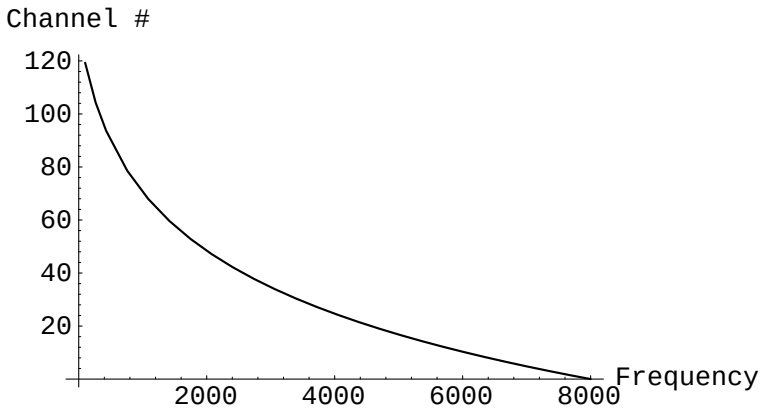
$$-\left(\frac{\text{EarQ Log[cf + EarQ minBW]}}{\text{stepfactor}}\right) + \frac{\text{EarQ Log[fn + EarQ minBW]}}{\text{stepfactor}}$$

```
N[index/.GlasbergBandwidthParms/.fn->8000/.stepfactor->.25/.
    cf->1000]
```

```
70.4687
```

We can use this equation to find how many channels are needed to cover an arbitrary low frequency sound. Let's plot this, assuming a Nyquist rate of 8000Hz. We overlap the filters by 25% (*stepfactor*=.25).

```
Plot[index/.GlasbergBandwidthParms/.
            stepfactor->.25/.fn->8000,
        {cf,100,8000},
        AxesLabel->{"Frequency","Channel #"}];
```



We need to invert this equation to find the mapping between index and center frequency. This will allow us to calculate the appropriate center frequency for any stage in the filter bank.

```
cfsoln = Simplify[Solve[index == i, cf][[1,1]]]
```

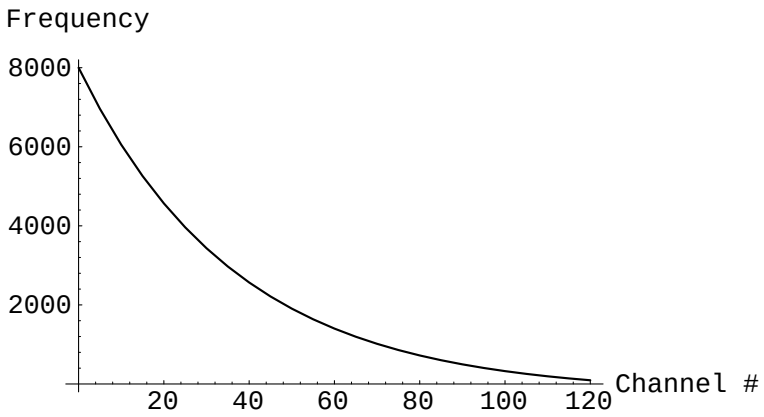
$$\text{cf} \to -(\text{EarQ minBW}) + \frac{\text{fn + EarQ minBW}}{E^{(i\ \text{stepfactor})/\text{EarQ}}}$$

```
N[cf/.cfsoln/.GlasbergBandwidthParms/.fn->8000/.stepfactor->.25/.
    i->70]
```

```
1015.64
```

We plot this solution to make sure it looks reasonable.

```
Plot[cf/.cfsoln/.GlasbergBandwidthParms/.
        fac->1.0/.stepfactor->.25/.fn->8000,{i,0,120},
        AxesLabel->{"Channel #","Frequency"}];
```



Alternatively, we might know how many channels we want within a frequency range. We can solve the *index* equation for the proper *stepfactor* to use.

```
stepSoln=Simplify[Solve[index==numChannels/.
    cf->lowFreq,stepfactor]]
```

$$\{\{\text{stepfactor} \to \frac{\text{EarQ}\ (\text{Log}[\text{fn} + \text{EarQ minBW}] - \text{Log}[\text{lowFreq} + \text{EarQ minBW}]}{\text{numChannels}}\}$$

To make this even easier to use, we can substitute this expression for the proper *stepfactor* into the equation for the center frequency. This gives us

```
fixedChannelCF=Simplify[cfsoln/.stepSoln]
```

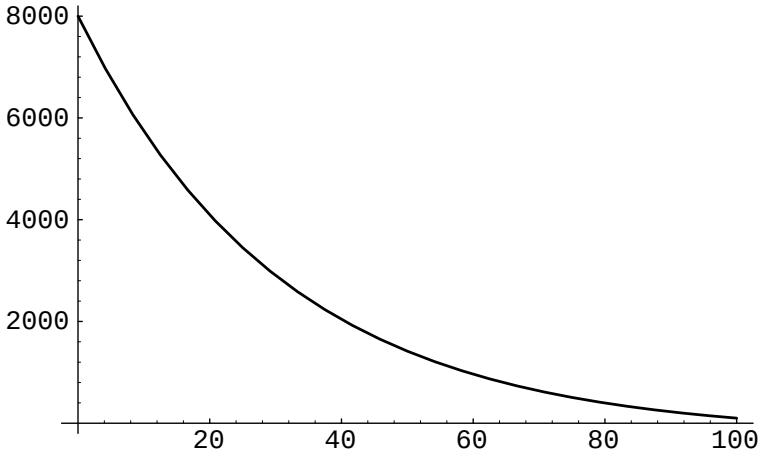
```
{cf -> -(EarQ minBW) + E
      (i (-Log[fn + EarQ minBW] + Log[lowFreq + EarQ minBW]))/numChan

       (fn + EarQ minBW)}
```

```
cf/.fixedChannelCF/.GlasbergBandwidthParms/.fn->8000/.
    numChannels->100/.lowFreq->100/.i->59
```

```
1002.3
```

Note that channel 0 is at the Nyquist rate. We should probably use channels 1 through *numChannels* when we design the filter bank. The plot below shows the center frequency for each of the 101 channels that range from the Nyquist rate (8000Hz) to 100Hz.

```
Plot[cf/.fixedChannelCF/.GlasbergBandwidthParms/.
    numChannels->100/.lowFreq->100/.fn->8000,
    {i,0,100}];
```



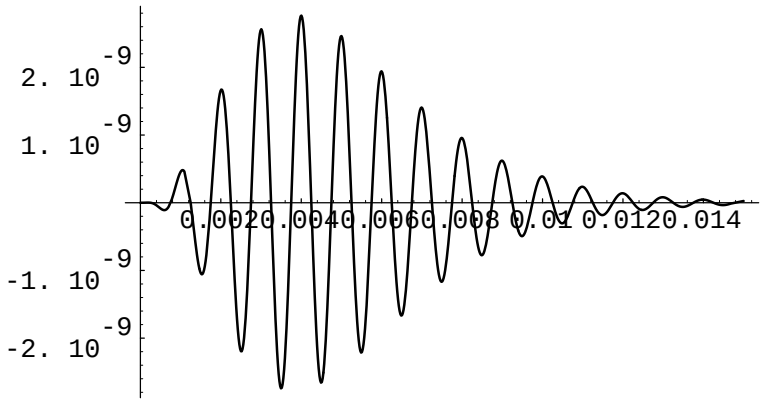
3.0 Gammatone Filters

Roy Patterson's ear model is based on impulse responses of the form

$$g[t] = \frac{a\, t^{n-1}\, \mathrm{Cos}[2\,\mathrm{Pi}\,fc\,t + phi]}{E^{2\,\mathrm{Pi}\,b\,t}}$$

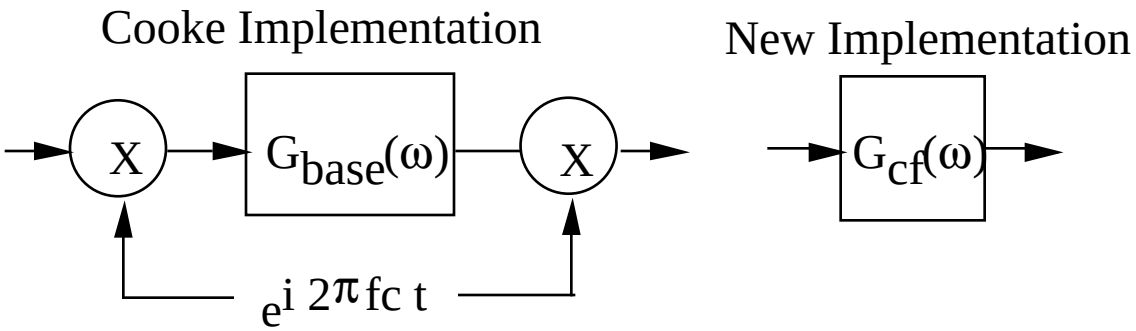
Here is the expected impulse response for a cochlear channel centered at 1000Hz (fc), a bandwidth (b) of 125Hz, and order (n) 4.

```
Plot[t^(4-1)E^(-2 Pi 125 t)Cos[2 Pi 1000 t],{t,0,.015}];
```



Patterson recommends setting the bandwidth, b, in the Gammatone filter to 1.019 times the ERB.

Martin Cooke [Cooke1991] describes an implementation that is similar to the one developed here, but not quite as efficient. Cooke develops a digital filter for the baseband Gammatone filter using the impulse invariance transformation. The original sound is first multiplied by a complex exponential at the desired center frequency, then filtered with a baseband Gammatone filter, then post-multiplied by the same exponential. The filter is a fourth order filter but with complex data so the net computational effort for the filter is the same as an eighth order filter. The filter described in this report is an eighth order filter, but doesn't need the pre- and post-modulations by a complex exponential. This difference is shown in the following figure. In both implentations, the cost of G(ω) is equivalent to an eighth order digital filter.



Section 3 of this report describes the properties and Laplace Transform of a Gammatone filter. First, in Section 3.1, the Laplace transform of the Gammatone filter is derived. In Section 3.2 the properties of this filter are explored. Sections 3.3 and 3.4 show how to convert the continuous Gammatone filter into a digital filter. Section 3.3 shows a numerical example while 3.4 shows the more general symbolic result. Finally, Section 3.5 shows a simplied version of the Gammatone filter without the zeros in the original filter. This all-pole version will be approximately half the computational cost of the exact formulation but will not be quite as sharp on the low frequency side of the resonance.

3.1 Laplace Transform

It would be nice if we could implement a channel of this ear model as a simple digital filter instead of the frequency-domain methods that have been used in the past. The damped exponential looks particularly easy to deal with. First let's recall a few Laplace transforms:

$$F[s] = \text{Integrate}\left[\frac{f[t]}{E^{t\ s}}, \{t, 0, \text{Inf}\}\right]$$

$$f[t] \to F[s]$$

$$t\ f[t] \to -F'[s]$$

$$t^n\ f[t] \to (-1)^n\ D[F[s], \{s, n\}]$$

In *Mathematica* notation, **D[F[s],{s,n}]** indicates the n-th derivative of the function *F[s]* with respect to variable *s*. Finally, the following Laplace tranform pair is used [CRC1966].

$$\frac{\text{Cos}[w\ t]}{E^{B\ t}} \to \frac{s + B}{(s + B)^2 + w^2}$$

Note that later we will replace B with 2πb or 2π 1.019 ERB(f) and w will be replaced with 2π cf. We ignore the optional phase term in the original Gammatone function because it is a minor effect which we do not feel is important for later processing. Now combining this identity with the Laplace transform of a function multiplied by t^n we get

$$\frac{t^n\ \text{Cos}[w\ t]}{E^{B\ t}} \to (-1)^n\ D\left[\frac{s + B}{(s + B)^2 + w^2}, \{s, n\}\right]$$

This is the basic form that we want. Let's see what derivatives do to the filter. According to Patterson [Patterson1992], we are mostly interested in the fourth order Gammatone filter, or n=4. Thus we are interested in the third derivative of the Gammatone function above. Here are the first three derivatives. We have ignored the factor of $(-1)^n$ because we will later normalize the gain of each filter.

$$\text{firstGammaLaplace} = \frac{s + B}{(s + B)^2 + w^2};$$

```
D[firstGammaLaplace,{s,1}]
```

$$\frac{-2\,(B+s)^2}{((B+s)^2+w^2)^2}+\frac{1}{(B+s)^2+w^2}$$

```
D[firstGammaLaplace,{s,2}]
```

$$\frac{8\,(B+s)^3}{((B+s)^2+w^2)^3}-\frac{6\,(B+s)}{((B+s)^2+w^2)^2}$$

```
filter = D[firstGammaLaplace,{s,3}]
```

$$\frac{-48\,(B+s)^4}{((B+s)^2+w^2)^4}+\frac{48\,(B+s)^2}{((B+s)^2+w^2)^3}-\frac{6}{((B+s)^2+w^2)^2}$$

We have saved this filter by storing it in a variable called *filter*. What does this look like when it is all put together? The *Mathematica* function **Together** will simplify the filter function.

```
rationalFilter = Together[filter]
```

$$(6\,(-B^4-4B^3s-6B^2s^2-4Bs^3-s^4+6B^2w^2+12Bsw^2+6s^2w^2-w^4))/(B^2+2Bs+s^2+w^2)^4$$

Ahh, this is interesting. The highest order of s in the numerator or the denominator is eight. Thus an eighth order filter can be used to implement the fourth order gammatone filter.

The expression in the denominator has two unique solutions, but each root is duplicated four times. Thus we have a set of four conjugate pole-pairs, all at the same location. Where are the zeros? We use the *Mathematica* function **Solve** to find the complex frequencies, **s**, where the numerator goes to zero.

```
zeros = Simplify[Solve[Numerator[rationalFilter]==0,s]]
```

```
{{s -> -B + Sqrt[3 + 2^3/2] w}, {s -> -B - Sqrt[3 + 2^3/2] w},
  {s -> -B + Sqrt[3 - 2^3/2] w}, {s -> -B - Sqrt[3 - 2^3/2] w}}
```

Which can be simplified to the following list (thanks to John Holdworth for pointing this out):

```
{{s -> -B + (1 + Sqrt[2]) w}, {s -> -B - (1 + Sqrt[2]) w},
  {s -> -B + (-1 + Sqrt[2]) w}, {s -> -B - (-1 + Sqrt[2]) w}}
```

The *Mathematica* function **N** is used to convert the roots in the list above into floating point numbers. We lose precision when we do this, so we save this for last.

```
N[zeros]
```

```
{{s -> -1. B + 2.41421 w}, {s -> -1. B - 2.41421 w},
  {s -> -1. B + 0.414214 w}, {s -> -1. B - 0.414214 w}}
```

This is a constellation of zeros around the point –B. Two zeros move left and right along the real axis by 2.414w, two other zeros move along the real axis by .414w.

It is interesting to note that if we define the impulse response with a sine instead of a cosine that one of the zeros disappear. The intial Laplace transform has two poles and no zeros. As we take each derivative in the Laplace domain (equivalent to multiplying the impulse response by t) we gain an extra zero. The resulting impulse response has just three zeros. We will expand on this in a later section.

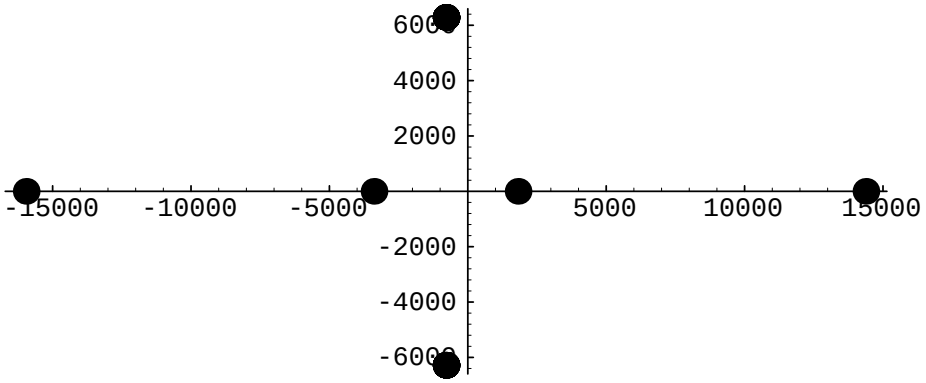
```
firstSineGammaLaplace=B/((s+B)^2+w^2);

Solve[Numerator[D[firstSineGammaLaplace,{s,3}]]==0,s]

{{s -> -B}, {s -> -B + w}, {s -> -B - w}}
```

We can solve a similar equation to find the roots of the Gammatone filter denominator. In this case there are four sets of complex-conjugate pole pairs. We put them in a list called **poles** for later reference.

```
poles = Simplify[Solve[Denominator[rationalFilter]==0,s]]

{{s -> -B + I w}, {s -> -B - I w}, {s -> -B + I w}, {s -> -B - I w},
  {s -> -B + I w}, {s -> -B - I w}, {s -> -B + I w}, {s -> -B - I w}}
```

We can now plot the location of these poles and zeros assuming a gammatone filter with a bandwidth of 125Hz and a center frequency of 1000Hz (this approximates a 1000Hz cochlear channel.) In the graph below, the four zeros are shown along the real (horizontal) axis. There are four sets of poles at each of the dots indicated near ±6000j (which indicates a resonance near 1000Hz.)

```
PlotPolesWithDots[complexpolelist_]:=
    ListPlot[Map[{Re[#],Im[#]}&,complexpolelist],
        Prolog->{AbsolutePointSize[10]},
        AspectRatio->Automatic,
        DisplayFunction->Identity];

Show[{PlotPolesWithDots[N[Map[Part[#,1,2]&,poles]/.
            B->2 Pi 125/.w->2 Pi f/.f->1000]],
    PlotPolesWithDots[N[Map[Part[#,1,2]&,zeros]/.
            B->2 Pi 125/.w->2 Pi f/.f->1000]]},
        DisplayFunction->$DisplayFunction];
```



■ 3.2 Signal Processing

This section analyses the continuous Gammatone filter and shows some of its properties. We will use an ERB filter centered at 1000 Hz to illustrate the following two sections. First, let's load a package of filter design and signal processing routines [Slaney1993].

```
<<FilterDesign
```

We can define a filter function, *F[s, cf]*, which is the desired ERB filter as a function of complex frequency *s*, and desired characteristic frequency *cf*.

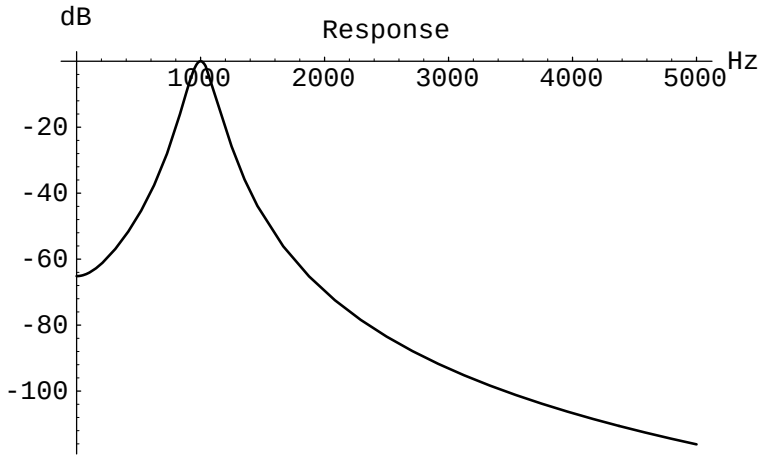
```
F[s_, cf_] := Release[rationalFilter/.
                B->(2 Pi 1.019 ERB[cf,EarQ,minBW,order])/.
                GlasbergBandwidthParms/.
                w->2 Pi cf]
```

```
N[F[s, 1000]]
```

$$(6.\ (-1.38824\ 10^{15} + 3.99865\ 10^{11}\ s + 2.32543\ 10^{8}\ s^2 - 3396.92\ s^3 - 1.\ s^4))\ /\ (4.01996\ 10^{7} + 1698.46\ s + s^2)^4$$

Let's plot the filter's response. We normalize the filter by its response at 1000Hz so that we get a dB scale in terms of the maximum response.

```
ContinuousFreqResponse[F[s,1000]/
                       ContinuousFilterMag[F[s,1000],1000],
                       5000];
```



Just to make sure, let's find the 3dB points of this filter. The **FilterSearch** routine finds the frequency that gives an arbitrary response. First, look for the point that is 3dB down in the octave (500->1000 Hz) below the CF, and then find it in the octave above. After determining the 3dB points, the filter's quality factor can be calculated.

```
lower3dB = FilterSearch[F[s,1000],
                  N[ContinuousFilterMag[F[s,1000],1000]
                              *Sqrt[2]/2],
                      500,1000]
```

```
941.209
```

```
upper3dB = FilterSearch[F[s,1000],
                  N[ContinuousFilterMag[F[s,1000],1000]
                              *Sqrt[2]/2],
                      1000,2000]
```

```
1058.79
```

The bandwidth of this filter is equal to

```
upper3dB - lower3dB
```

```
117.58
```

Finally, the Q_{3dB} is given by

```
1000/(upper3dB - lower3dB)
```

```
8.50485
```

The ERB here is close to the 3dB bandwidth.

```
ERB[1000,EarQ,minBW, order]/.GlasbergBandwidthParms
```

```
132.639
```

The filter design package [Slaney1993] wants to represent filters in terms of a structure called the GZP, or Gain-Zeros-Poles. These lists are easier to work with than polynomial equations and are simply a list of poles and zeros of the filter. The **Map** function allows us to pick apart the solutions stored in the poles and zeros variables computed in Section 3.1. The gain term, first element in the list, is equal to six over the gain of the original F[s] filter at 1000Hz. We want to normalize the filter so it has unity gain and the six comes from the original *rationalFilter* definition.

```
GZP = List[6/N[ContinuousFilterMag[F[s,1000],1000]],
            Map[Last,zeros,2],
            Map[Last,poles,2]]/.w->2 Pi cf
```

```
{1.04021 10^12, {-B + 2 Sqrt[3 + 2^3/2] cf Pi,
   -B - 2 Sqrt[3 + 2^3/2] cf Pi, -B + 2 Sqrt[3 - 2^3/2] cf Pi,
   -B - 2 Sqrt[3 - 2^3/2] cf Pi},
  {-B + 2 I cf Pi, -B - 2 I cf Pi, -B + 2 I cf Pi, -B - 2 I cf Pi,
   -B + 2 I cf Pi, -B - 2 I cf Pi, -B + 2 I cf Pi, -B - 2 I cf Pi}}
```

We compute the exact numerical version of this GZP structure, this time assuming a 1000Hz filter, so that we have exact numbers to talk about. We'll plot the resulting filter, just to make sure that we have still have a bandpass filter at 1000Hz.

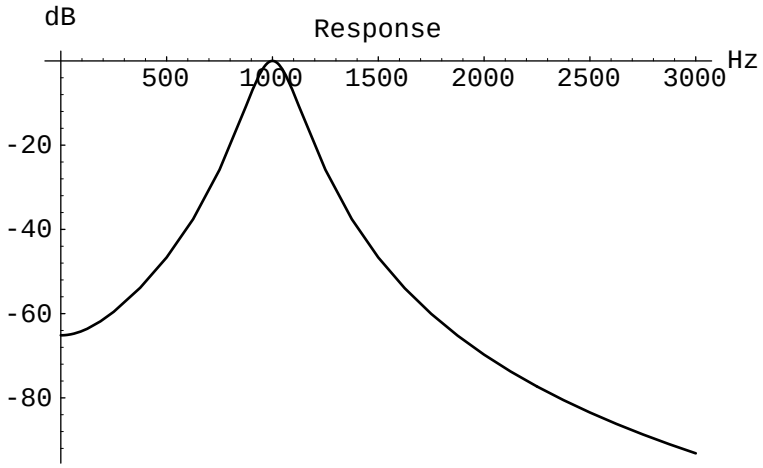
```
numGZP = N[GZP/.B->2 Pi 1.019 ERB[cf,EarQ,minBW,order]/.
           GlasbergBandwidthParms/.cf->1000]
```

```
{1.04021 10^12, {14319.7, -16018.2, 1753.35, -3451.81},
  {-849.23 + 6283.19 I, -849.23 - 6283.19 I, -849.23 + 6283.19 I,
   -849.23 - 6283.19 I, -849.23 + 6283.19 I, -849.23 - 6283.19 I,
   -849.23 + 6283.19 I, -849.23 - 6283.19 I}}
```

```
FilterFromGZP[numGZP]
```

$$\frac{1.04021\ 10^{12}\ (-14319.7 + s)\ (-1753.35 + s)\ (3451.81 + s)\ (16018.2 + s}{(849.23 - 6283.19\ I + s)^4\ (849.23 + 6283.19\ I + s)^4}$$

```
ContinuousFreqResponse[FilterFromGZP[numGZP],3000];
```



■ 3.3 Numerical Z-Transform

We can implement the fourth-order Gammatone impulse response as a cascade of four second-order filters. The analysis to this point is in the Laplace, or continuous domain. We need to convert these filters into their equivalent sampled form for use in a program. This section describes this conversion using the numerical example shown in the previous section; Section 3.4 describes the more general symbolic method.

An easy way to convert a continuous (analog) filter into its digital equivalent is to use the impulse invariance design technique. The impulse invariance technique maps a continuous filter into an equivalent sampled version by matching the impulse response in the time domain. But, due to aliasing, this might not be the most accurate result. Cooke's report addresses several alternatives and concludes that the impulse invariance technique is most accurate for baseband gammatone filters because there is very little aliasing. In our case, since the desired filter responses are defined in terms of their impulse response, the impulse invariance design technique seems the most natural.

The impulse invariance filter transformation requires that we expand the filter into its partial fractions expansion, and then replace each pole by its discrete equivalent. A single pole has a

simple impulse response (it's just a decaying exponential) so the equivalent digital filter is a single discrete pole.

The final digital filter will be a cascade of four biquadratic sections per cochlear channel. This costs us 20 multiplies and 12 adds per channel.

Unfortunately, we can't do a simple partial fractions expansion because we have multiple poles at the same location. Instead we split the filter into a cascade of four biquadratic sections. We first group the poles into complex-conjugate pairs. We will form four filters by taking part of the gain, two of the zeros, and a pair of poles. Since we have four zeros, we'll assign one zero to each of the four filters. A continuous filter with a complex root is transformed into a digital filter with complex coefficients. But by combining the result of transforming two complex conjugate poles, the resulting digital filter has real coefficients.

```
conjugatePoles = GroupRoots[numGZP[[3]]]
```

```
{{-849.23 + 6283.19 I, -849.23 - 6283.19 I},
  {-849.23 - 6283.19 I, -849.23 + 6283.19 I},
  {-849.23 + 6283.19 I, -849.23 - 6283.19 I},
  {-849.23 - 6283.19 I, -849.23 + 6283.19 I}}
```

```
gammaZeros = numGZP[[2]]
```

```
{14319.7, -16018.2, 1753.35, -3451.81}
```

We really should keep the gain in each filter, perhaps putting 1/4 of the gain into each filter. But this makes the numbers weird, and make it hard to see the right answer. Let's arbitrarily set the gain of each of the following four filters to 1. Later we will adjust the gain numerically, if needed. Each filter has two conjugate poles and a single zero along the real axis.

```
filter1 = List[1, List[gammaZeros[[1]]],
                  conjugatePoles[[1]]]
filter2 = List[1, List[gammaZeros[[2]]],
                  conjugatePoles[[2]]]
filter3 = List[1,List[gammaZeros[[3]]],
                  conjugatePoles[[3]]]
filter4 = List[1,List[gammaZeros[[4]]],
                  conjugatePoles[[4]]]
```

```
{1, {14319.7}, {-849.23 + 6283.19 I, -849.23 - 6283.19 I}}
```

```
{1, {-16018.2}, {-849.23 - 6283.19 I, -849.23 + 6283.19 I}}
```

```
{1, {1753.35}, {-849.23 + 6283.19 I, -849.23 - 6283.19 I}}
```

```
{1, {-3451.81}, {-849.23 - 6283.19 I, -849.23 + 6283.19 I}}
```

For each filter, we use the **StandardZTransform** function to map the biquadratic (continuous domain) filter into the equivalent Z-domain filter. The z-domain filters can be implemented on a digital computer. We do this for each of the specific four filters we designed at 1000Hz to make sure we can do this properly. The **StandardZTransform** function converts a list of poles in GZP format into a rational function of the variable z.

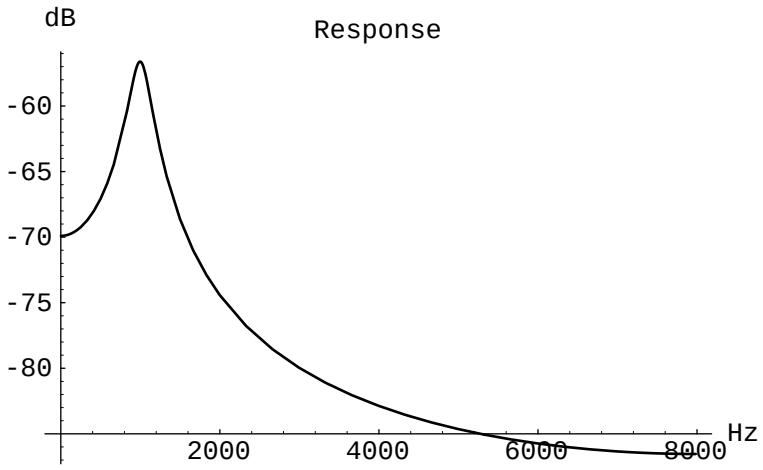
We use the **Simplify** function to put the equations in the canonical form.

```
zfilt1 = Simplify[StandardZTransform[filter1, 1/16000]]
```

$$\frac{0.0000625\ (-1.75224 + z)\ z}{0.899286 - 1.75224\ z + z^2}$$

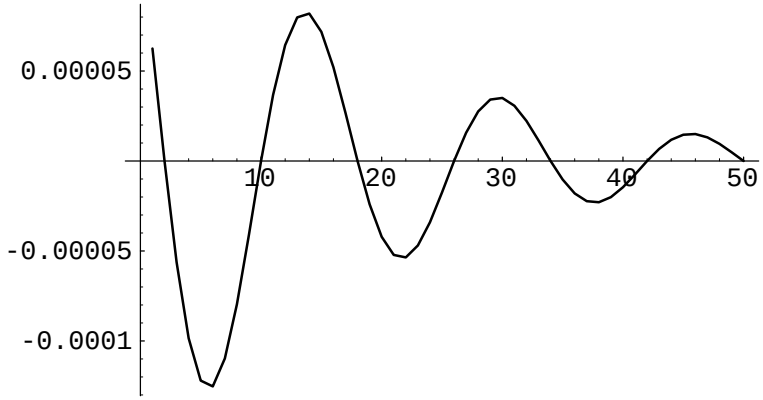
The frequency response of this filter is found by substituting $e^{i\,2\pi\,f/fs}$ for z in the above equation with f set to the frequency of interest and fs set to the digital sampling rate (16000 in this example)

```
FreqResponse[zfilt1,16000];
```



Finally, we can use the **FindImpulseResponse** function to digitally simulate the resulting filter and show its response.

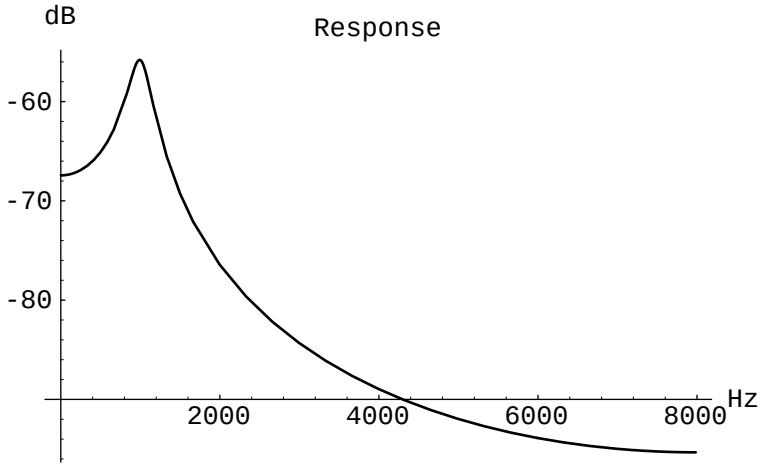
```
ListPlot[FindImpulseResponse[zfilt1,50],PlotJoined->True];
```



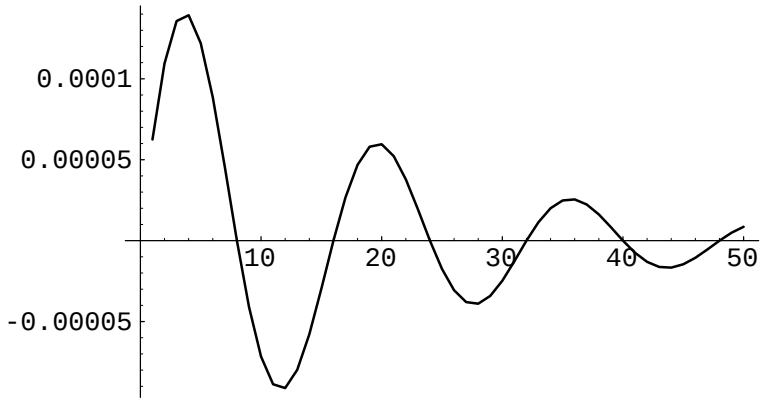

```
zfilt2 = Simplify[StandardZTransform[filter2, 1/16000]]
```

$$\frac{0.0000625\ z^2}{0.899286 - 1.75224\ z + z^2}$$

```
FreqResponse[zfilt2,16000];
```



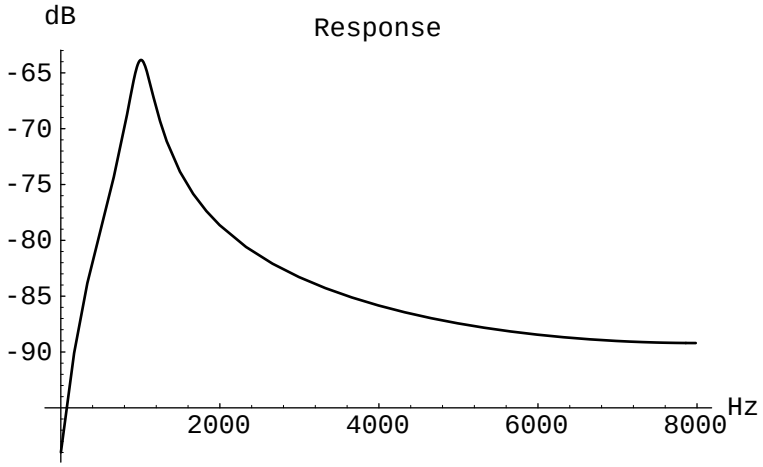

```
ListPlot[FindImpulseResponse[zfilt2,50],PlotJoined->True];
```



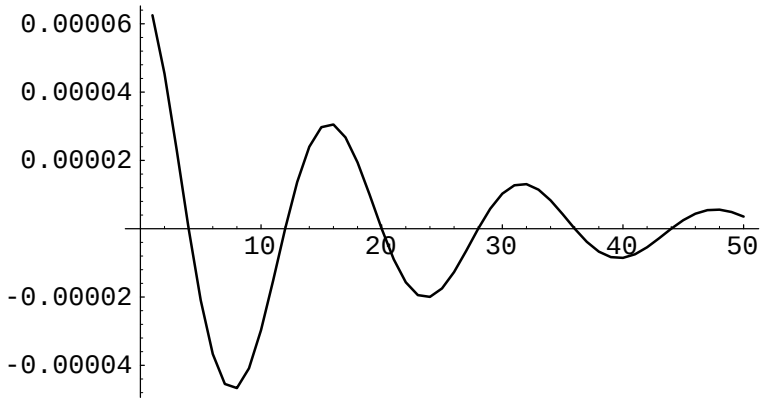

```
zfilt3 = Simplify[StandardZTransform[filter3, 1/16000]]
```

$$\frac{0.0000625\ (-1.02644 + z)\ z}{0.899286 - 1.75224\ z + z^2}$$

```
FreqResponse[zfilt3,16000];
```



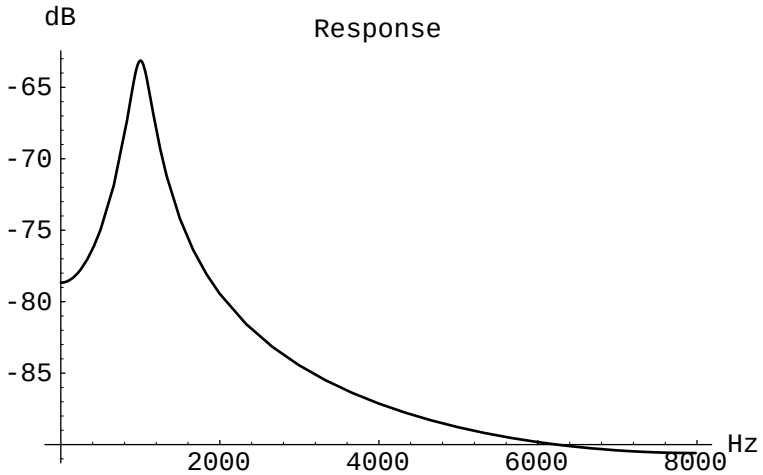

```
ListPlot[FindImpulseResponse[zfilt3,50],PlotJoined->True];
```



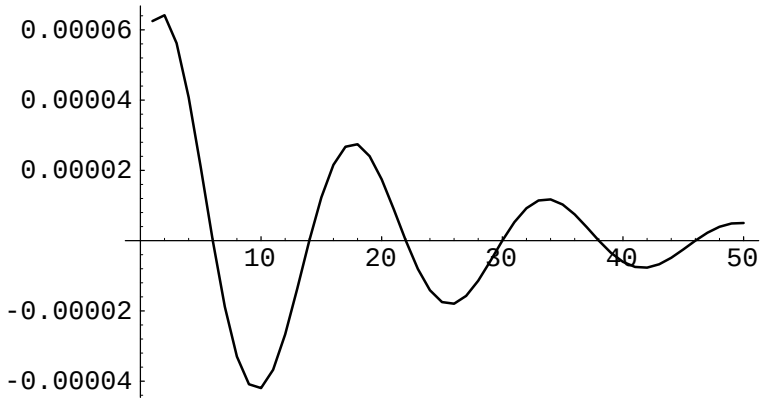

```
zfilt4 = Simplify[StandardZTransform[filter4,1/16000]]
```

$$\frac{0.0000625\ (-0.725803 + z)\ z}{0.899286 - 1.75224\ z + z^2}$$

```
FreqResponse[zfilt4,16000];
```

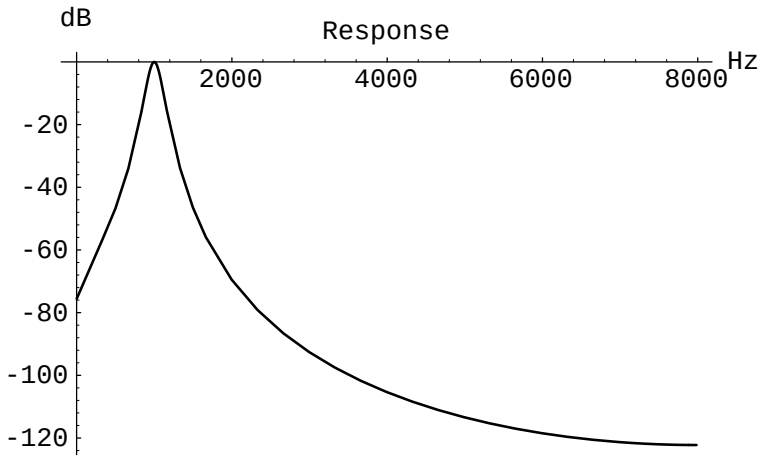



```
ListPlot[FindImpulseResponse[zfilt4,50],PlotJoined->True];
```



Finally, we can plot the response for the cascade of all four filters.

```
FreqResponse[Release[zfilt1 zfilt2 zfilt3 zfilt4/
        FilterMag[zfilt1 zfilt2 zfilt3 zfilt4,1000, 16000]], 16000]
```



3.4 Symbolic Z-Transform

This section derives the digital filter that approximates a Gammatone function with center frequency w=2π fc and a bandwidth of B radians. The symbolic Laplace transform for this function was derived in Section 3.1.

Unfortunately, the **StandardZTransform** code used in Section 3.3 needs numeric values for the poles so they can be sorted and conjugate pole pairs can be found. But in this case we have a symbolic expression for the conjugate pole pair. So, instead let's write a simple function that performs the transformation. Recall, we first expand the transfer function into its partial fractions expansion. This gives us an expression for the filter stage in the following form.

$$\frac{r1}{s - p1} + \frac{r2}{s - p2}$$

In the partial fractions expansion, *r1* and *r2* are the residues of the original filter function when evaulated at *p1* and *p2*. We find the z-transform of the equivalent impulse invariant filter, assuming a sampling interval of *T*, by replacing the simple s-domain pole with a modified z-domain pole.

$$\frac{r}{s - p} \rightarrow \frac{T\,r}{1 - \frac{E^{T\,p}}{z}} = \frac{T\,r\,z}{z - E^{T\,p}}$$

The following function is used to make this transformation. The variables *r1* and *r2* hold the two residues. The local variable *filt* is used to hold the filter, and a number of simplifications are performed to reduce the result into its canonical form.

```
SymbolicZTransform[{gain_, zeros_, {p1_,p2_}}, T_] :=
    Block[{r1, r2, filt},
        r1 = Simplify[EvaluateGZP[gain, zeros, {p2}, p1]];
        r2 = Simplify[EvaluateGZP[gain, zeros, {p1}, p2]];
        (* Print["Residue 1 is ", r1];                    *)
        (* Print["Residue 2 is ", r2];                    *)
        filt = Together[(T r1)/(1-E^(T p1)/z) +
                                  (T r2)/(1-E^(T p2)/z)];
        filt = Collect[Simplify[ComplexExpand[Numerator[filt]]],z]/
                 Collect[Simplify[ComplexExpand[Denominator[filt]]]
        Return[filt]
        ]
```

We first test it on a numeric filter we've already seen. Within the limits of floating point accuracy, we get the same result:

```
SymbolicZTransform[filter1, 1/16000]
```

$$\frac{-0.000109515\ z + 0.0000625\ z^2}{0.899286 - 1.75224\ z + z^2}$$

Here's the real result. Let's take the first two zeros and two of the conjugate poles and compute the equivalent impulse invariant filter.

```
zfilt1 = SymbolicZTransform[{1,{GZP[[2]][[1]]},
                             {GZP[[3]][[1]], GZP[[3]][[2]]}},T]
```

$$\frac{-2\,T\,z^2 + z\left(\frac{2\,T\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}} + \frac{2\,\text{Sqrt}[3 + 2^{3/2}]\,T\,\text{Sin}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)}{\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}}$$

```
zfilt2 = SymbolicZTransform[{1,{GZP[[2]][[2]]},
                             {GZP[[3]][[1]], GZP[[3]][[2]]}},T]
```

$$\frac{-2\,T\,z^2 + z\left(\frac{2\,T\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}} - \frac{2\,\text{Sqrt}[3 + 2^{3/2}]\,T\,\text{Sin}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)}{\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}}$$

```
zfilt3 = SymbolicZTransform[{1,{GZP[[2]][[3]]},
                             {GZP[[3]][[1]], GZP[[3]][[2]]}},T]
```

$$\frac{-2\,T\,z^2 + z\left(\frac{2\,T\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}} + \frac{2\,\text{Sqrt}[3 - 2^{3/2}]\,T\,\text{Sin}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)}{\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}}$$

```
zfilt4 = SymbolicZTransform[{1,{GZP[[2]][[4]]},
                             {GZP[[3]][[1]], GZP[[3]][[2]]}},T]
```

$$\frac{-2\,T\,z^2 + z\left(\frac{2\,T\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}} - \frac{2\,\text{Sqrt}[3 - 2^{3/2}]\,T\,\text{Sin}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)}{\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}}$$

This is the completely general result. We can plug in any sampling interval and any center frequency to get the filter somewhere along the Basilar Membrane. We set the *cf* to 1000 and the sampling interval (*T*) to 1/16000 so we can compare it to the result above.

```
Simplify[N[zfilt3/.B->(2 Pi 1.019 ERB[cf,EarQ,minBW,order])/.
                      GlasbergBandwidthParms/.
                      cf->1000/.
                      T->1/16000]]
```

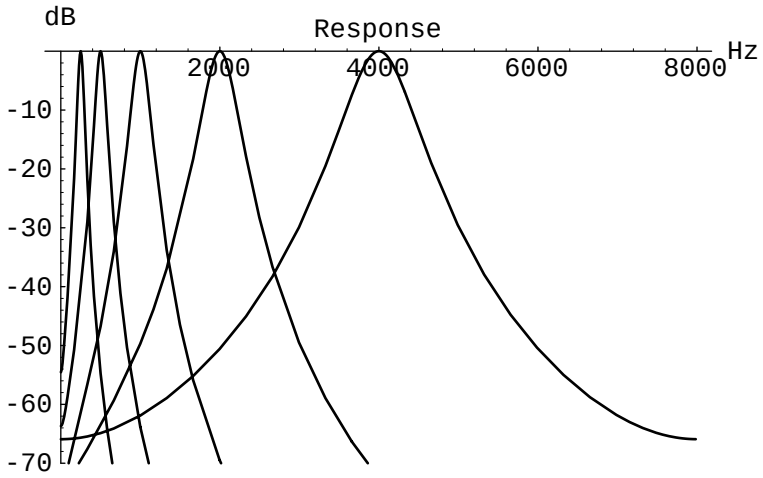
$$\frac{0.0000625\ (-1.02644 + z)\ z}{0.899286 - 1.75224\ z + z^2}$$

Now, to create a family of frequency responses, we would like to have a little function that puts everything together, simplifies, and adjusts the gain. This function is called **MakeERBFilter** and takes two arguments, the center frequency and the sampling rate, *fs*.

```
MakeERBFilter[centerfreq_, fs_] :=
    Block[{filter, gain},
        filter = Simplify[zfilt1 zfilt2 zfilt3 zfilt4/.
                          B->(2 Pi 1.019 ERB[cf,EarQ,minBW,order])/
                          GlasbergBandwidthParms/.cf->centerfreq/.
                          T->1/fs];
        filter = Simplify[N[filter]];
        gain = N[FilterMag[filter, centerfreq, fs]];
        Return[filter/gain]
    ]
```

Let's plot a few of these digital Gammatone filters. Here's the result. We limit the dB range from {-70 to 0} to keep the plot looking reasonable.

```
Show[Table[FreqResponse[MakeERBFilter[125*2^i,16000], 16000,
                      {PlotRange->{-70,0},
                       DisplayFunction->Identity}],
            {i,5}],
    DisplayFunction->$DisplayFunction];
```



■ 3.5 All-Pole Gammatone Approximation

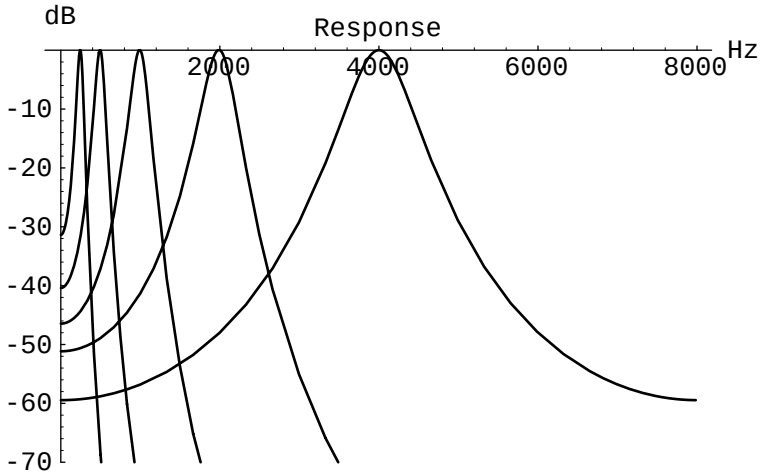
All of the zeros in the Gammatone filter, *rationalFilter*, are on the real axis. This means that the zeros have a small effect near the center frequency of each filter. By ignoring the zeros in the original filter we cut the computation effort nearly in half, at some reduction in the filter's attenuation near DC. This means that we have four second order sections, each with the identical set of conjugate poles near the resonant frequency.

```
twopolefilt = SymbolicZTransform[{1,{},
                          {GZP[[3]][[1]], GZP[[3]][[2]]}},T]
```

$$\frac{-2\,T\,z\,\mathrm{Sin}[2\,cf\,\pi\,T]}{E^{B\,T}\left(\frac{-4\,cf\,\pi}{E^{2\,B\,T}} - 4\,cf\,\pi\,z^2 + \frac{8\,cf\,\pi\,z\,\mathrm{Cos}[2\,cf\,\pi\,T]}{E^{B\,T}}\right)}$$

We can now plot the response for a cascade of four of these second order sections. The bandwidth is similar to the exact solution but now there is less attenuation at DC.

```
MakePoleERBFilter[centerfreq_, fs_] :=
    Block[{filter, gain},
        filter = Simplify[twopolefilt twopolefilt twopolefilt
                    twopolefilt/.
                        B->(2 Pi 1.019 ERB[cf,EarQ,minBW,order])/
                        GlasbergBandwidthParms/.cf->centerfreq/.
                        T->1/fs];
        filter = Simplify[N[filter]];
        gain = N[FilterMag[filter, centerfreq, fs]];
        Return[filter/gain]
    ]

Show[Table[FreqResponse[MakePoleERBFilter[125*2^i,16000], 16000,
                    {PlotRange->{-70,0},
                     DisplayFunction->Identity}],
            {i,5}],
    DisplayFunction->$DisplayFunction];
```



Some versions of *Mathematica* include a package that performs Laplace Transforms. We use this package to derive the impulse response of the all-pole approximation to the fourth order Gammatone filter. We invert four copies of the two conjugate poles described in Section 3.1.

```
<<LaplaceTransform.m

InverseLaplaceTransform[1/((B+s)^2+w^2)^4,s,t]
```

$$\frac{5\,I}{32}\,\frac{E^{-I\,t\,(-I\,B+w)}}{w^7} - \frac{5\,I}{32}\,\frac{E^{I\,t\,(I\,B+w)}}{w^7} -$$

$$\frac{5\,E^{-I\,t\,(-I\,B+w)}\,t}{32\,w^6} - \frac{5\,E^{I\,t\,(I\,B+w)}\,t}{32\,w^6} -$$

$$\frac{I}{16}\,\frac{E^{-I\,t\,(-I\,B+w)}\,t^2}{w^5} + \frac{I}{16}\,\frac{E^{I\,t\,(I\,B+w)}\,t^2}{w^5} +$$

$$\frac{E^{-I\,t\,(-I\,B+w)}\,t^3}{96\,w^4} + \frac{E^{I\,t\,(I\,B+w)}\,t^3}{96\,w^4}$$

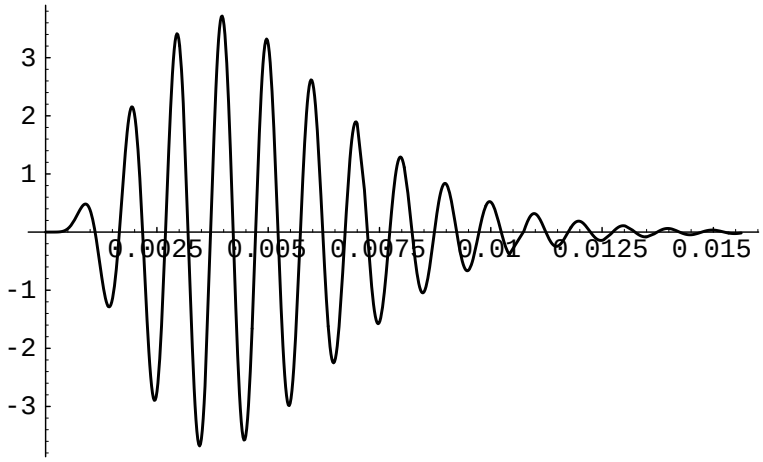
We can't convince *Mathematica* to simplify this expression, but if we combine complex exponentials by hand we get the following equivalent form. Note that it is a function of the first four Gammatone filters.

$$\frac{5\,\text{Sin}[w\,t]}{16\,w^7} - \frac{5\,t\,\text{Cos}[w\,t]}{16\,w^6} - \frac{t^2\,\text{Sin}[w\,t]}{8\,w^5} + \frac{t^3\,\text{Cos}[w\,t]}{48\,w^4}$$

$$\frac{}{E^{B\,t}}$$

Finally, here is the resulting theoretical impulse response for a 1000Hz all-pole Gammatone with a 125Hz bandwidth.

```
Plot[Release[10^26*E^(-B*t)(5*Sin[w*t]/16/w^7-
            5*t*Cos[w*t]/16/w^6-t^2*Sin[w*t]/8/w^5+
            t^3*Cos[w*t]/48/w^4)/.
        B->2*Pi*125/.w->2*Pi*1000],{t,0,250/16000},
    PlotRange->All];
```



■ 3.6 Testing

These are some simple test results. Compare them to the output of other implementations to verify that your program is correct.. We'll compute the results for two different filter locations, the first channel and the (arbitrary) 23rd channel. For each channel we compute the center frequency and the four digital filters. We use Glasberg's function for the ERB, overlap each filter by 50%, and assume a sampling rate of 16000 Hz.

```
f=cf/.cfsoln/.GlasbergBandwidthParms/.stepfactor->.5/.
                fn->8000/.i->1
```

```
7567.67
```

```
N[zfilt1/.cf->f/.B->2*Pi*1.019*ERB[f,EarQ,minBW,order]/.
        GlasbergBandwidthParms/.T->1/16000]
```

$$\frac{-0.0000515667\ z - 0.000125\ z^2}{-1.01983 - 2.81527\ z - 2.\ z^2}$$

```
N[zfilt2/.cf->f/.B->2*Pi*1.019*ERB[f,EarQ,minBW,order]/.
        GlasbergBandwidthParms/.T->1/16000]
```

$$\frac{-0.000124388\ z - 0.000125\ z^2}{-1.01983 - 2.81527\ z - 2.\ z^2}$$

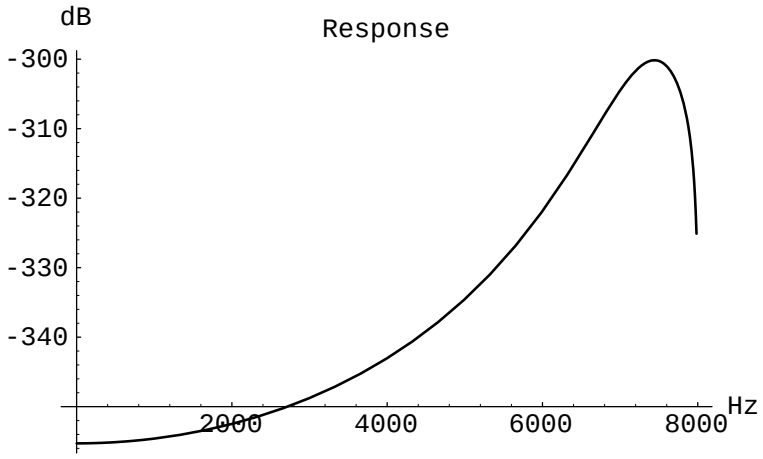
```
N[zfilt3/.cf->f/.B->2*Pi*1.019*ERB[f,EarQ,minBW,order]/.
        GlasbergBandwidthParms/.T->1/16000]
```

$$\frac{-0.0000817302\ z - 0.000125\ z^2}{-1.01983 - 2.81527\ z - 2.\ z^2}$$

```
N[zfilt4/.cf->f/.B->2*Pi*1.019*ERB[f,EarQ,minBW,order]/.
        GlasbergBandwidthParms/.T->1/16000]
```

$$\frac{-0.0000942243\ z - 0.000125\ z^2}{-1.01983 - 2.81527\ z - 2.\ z^2}$$

```
FreqResponse[zfilt1 zfilt2 zfilt3 zfilt4 /.
                cf->f/.B->
                2*Pi*1.019*ERB[f,EarQ,minBW,order]/.T->1/16000/.
                GlasbergBandwidthParms,
            16000];
```



Now, let's look at the 23rd channel.

```
f=cf/.cfsoln/.GlasbergBandwidthParms/.stepfactor->.5/.
                fn->8000/.i->23
```

2149.37

```
N[zfilt1/.cf->f/.B->2*Pi*1.019*ERB[f,EarQ,minBW,order]/.
        GlasbergBandwidthParms/.T->1/16000]
```

$$\frac{0.000278461\ z - 0.000125\ z^2}{-1.62857 + 2.39829\ z - 2.\ z^2}$$

```
N[zfilt2/.cf->f/.B->2*Pi*1.019*ERB[f,EarQ,minBW,order]/.
        GlasbergBandwidthParms/.T->1/16000]
```

$$\frac{-0.000128568\ z - 0.000125\ z^2}{-1.62857 + 2.39829\ z - 2.\ z^2}$$

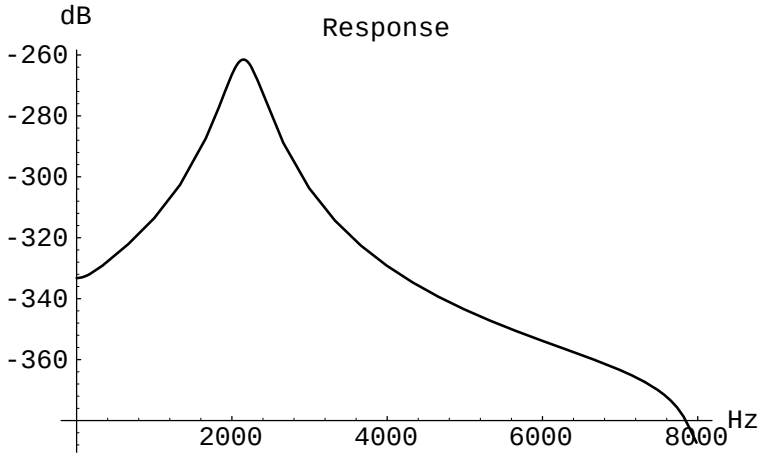
```
N[zfilt3/.cf->f/.B->2*Pi*1.019*ERB[f,EarQ,minBW,order]/.
        GlasbergBandwidthParms/.T->1/16000]
```

$$\frac{0.000109864\ z - 0.000125\ z^2}{-1.62857 + 2.39829\ z - 2.\ z^2}$$

```
N[zfilt4/.cf->f/.B->2*Pi*1.019*ERB[f,EarQ,minBW,order]/.
        GlasbergBandwidthParms/.T->1/16000]
```

$$\frac{0.0000400291\ z - 0.000125\ z^2}{-1.62857 + 2.39829\ z - 2.\ z^2}$$

```
FreqResponse[zfilt1 zfilt2 zfilt3 zfilt4 /.
                cf->f/.B->
                2*Pi*1.019*ERB[f,EarQ,minBW,order]/.T->1/16000/.
                GlasbergBandwidthParms,
            16000];
```



4.0 Implementation

This section of this report will describe two different implementations of the Gammatone filters. Section 4.1 will describe the implementation using second order filters. Section 4.2 will describe the implementation using the MATLAB digital signal processing environment.

4.1 Second-Order Section Implementation

We have designed four different digital filter stages. Together they implement a fourth order Gammatone filterbank as an eighth order digital filter. Each of these stages is a second order (biquadratic) digital filter. The four digital filters are shown below.

zfilt1

$$\frac{-2\,T\,z^2 + z\left(\frac{2\,T\,\cos[2\,cf\,\pi\,T]}{E^{B\,T}} + \frac{2\,\sqrt{3+2^{3/2}}\,T\,\sin[2\,cf\,\pi\,T]}{E^{B\,T}}\right)}{\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\cos[2\,cf\,\pi\,T]}{E^{B\,T}}}$$

zfilt2

$$\frac{-2\,T\,z^2 + z\left(\frac{2\,T\,\cos[2\,cf\,\pi\,T]}{E^{B\,T}} - \frac{2\,\sqrt{3+2^{3/2}}\,T\,\sin[2\,cf\,\pi\,T]}{E^{B\,T}}\right)}{\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\cos[2\,cf\,\pi\,T]}{E^{B\,T}}}$$

zfilt3

$$\frac{-2\,T\,z^2 + z\left(\frac{2\,T\,\cos[2\,cf\,\pi\,T]}{E^{B\,T}} + \frac{2\,\sqrt{3-2^{3/2}}\,T\,\sin[2\,cf\,\pi\,T]}{E^{B\,T}}\right)}{\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\cos[2\,cf\,\pi\,T]}{E^{B\,T}}}$$

zfilt4

$$\frac{-2\,T\,z^2 + z\left(\frac{2\,T\,\cos[2\,cf\,\pi\,T]}{E^{B\,T}} - \frac{2\,\sqrt{3-2^{3/2}}\,T\,\sin[2\,cf\,\pi\,T]}{E^{B\,T}}\right)}{\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\cos[2\,cf\,\pi\,T]}{E^{B\,T}}}$$

The total filter is given by the equation:

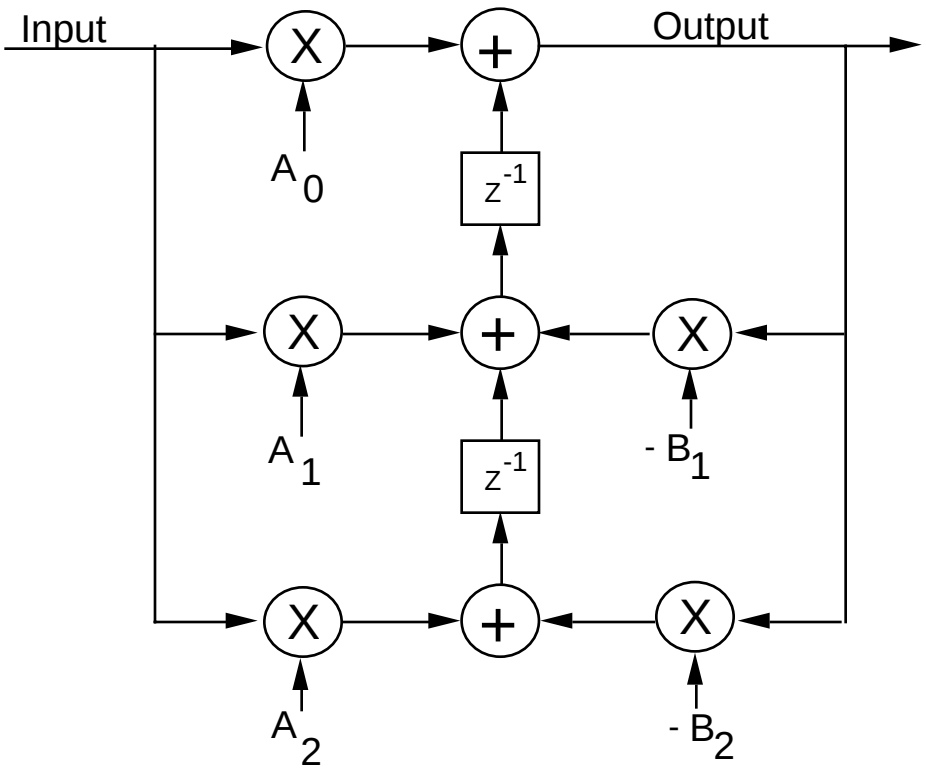
GammaFilt = (zfilt1 zfilt2 zfilt3 zfilt4)

$$\left((-2\,T\,z^2 + z\left(\frac{2\,T\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}} - \frac{2\,\text{Sqrt}[3 - 2^{3/2}]\,T\,\text{Sin}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)\right)$$
$$\left(-2\,T\,z^2 + z\left(\frac{2\,T\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}} + \frac{2\,\text{Sqrt}[3 - 2^{3/2}]\,T\,\text{Sin}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)\right)$$
$$\left(-2\,T\,z^2 + z\left(\frac{2\,T\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}} - \frac{2\,\text{Sqrt}[3 + 2^{3/2}]\,T\,\text{Sin}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)\right)$$
$$\left(-2\,T\,z^2 + z\left(\frac{2\,T\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}} + \frac{2\,\text{Sqrt}[3 + 2^{3/2}]\,T\,\text{Sin}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)\right) \Big/$$
$$\left(\frac{-2}{E^{2\,B\,T}} - 2\,z^2 + \frac{4\,z\,\text{Cos}[2\,cf\,\text{Pi}\,T]}{E^{B\,T}}\right)^4$$

To calculate any one channel of the Patterson filter bank we need to substitute appropriate values for the center frequency (*cf*), bandwidth (*b*), and sampling interval (*T*). The center frequency is determined by position along the Basilar Membrane and is a free parameter. In Patterson's model the bandwidth is fixed at (2π 1.019 ERB[cf]). The sampling interval is generally a function of the sound input system.

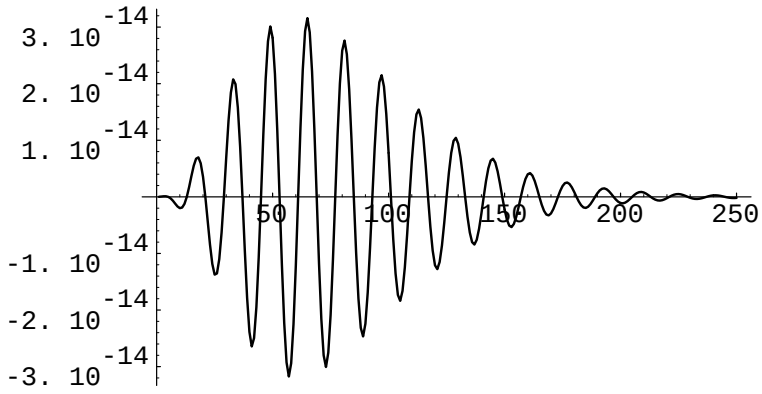
Each of the four filter stages is implemented with the following filter structure. After performing the indicated substitutions the 5 filter coefficients can be calculated and used to perform the digital filter. The two equivalent equations below are implemented by the digital filter diagrammed.

$$\frac{A0 + \frac{A1}{z} + \frac{A2}{z^2}}{1 + \frac{B1}{z} + \frac{B2}{z^2}} = \frac{A0\,z^2 + A1\,z + A2}{z^2 + B1\,z + B2}$$



Finally, here is the resulting time-domain impulse response for a fourth-order Gammatone filter, calculated by simulating the filter in the time-domain.

```
numGammaFilt = Simplify[N[GammaFilt/.B->2*Pi*125/.
               cf->1000/.
               T->1/16000]];
numGammaPoints = FindImpulseResponse[numGammaFilt,250];
ListPlot[numGammaPoints, PlotJoined->True];
```



We can also compare this result to the theoretical result.

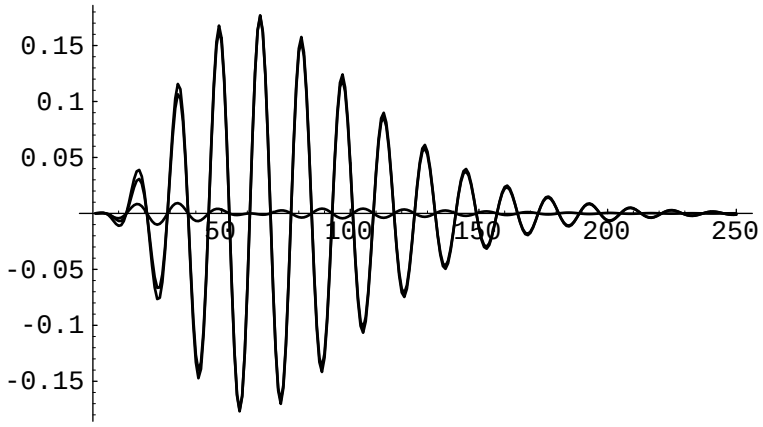
```
numGammaTheory = Table[N[t^(4-1)E^(-2 Pi 125 t)Cos[2 Pi 1000 t]/.
                        t->i/16000],{i,0,249}];
```

We normalize both the digitally simulated Gammatone impulse response, *numGammaPoints*, and the exact values, *numGammaTheory*, by dividing by their average RMS value.

```
numGammaPoints = numGammaPoints/
                        Sqrt[Apply[Plus,Map[#^2&,numGammaPoints]]];
numGammaTheory = numGammaTheory/
                        Sqrt[Apply[Plus,Map[#^2&,numGammaTheory]]];
```

We subtract the two lists of points to see the error. The following plot overlays the theoretical, the digital simulation, and the resulting error curves.

```
Show[{ListPlot[numGammaTheory,DisplayFunction->Identity,
                        PlotJoined->True],
          ListPlot[numGammaPoints,DisplayFunction->Identity,
                        PlotJoined->True],
          ListPlot[numGammaPoints-numGammaTheory,
                        PlotJoined->True, DisplayFunction->Identity]}
     DisplayFunction->$DisplayFunction, PlotRange->All];
```



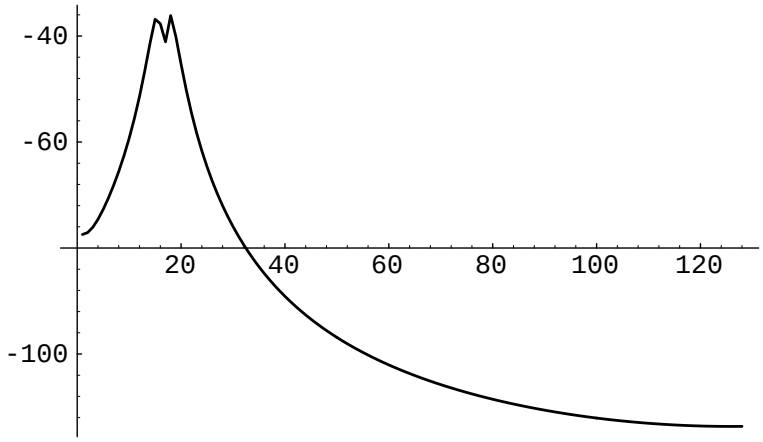
This error signal has an average (RMS) error of

```
Sqrt[Apply[Plus,Map[#^2&,numGammaPoints-numGammaTheory]]]
```

```
0.0435394
```

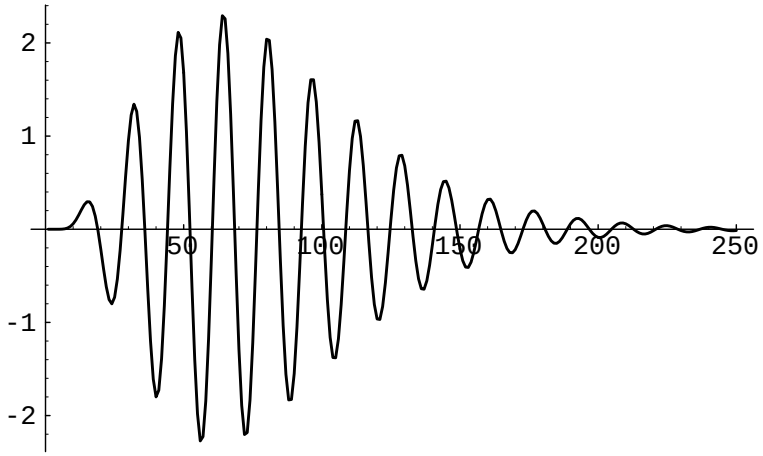
And the error is distributed across the frequency spectrum as shown in the following plot. We plot only the first half of the Fourier transform, from DC to the Nyquist frequency.

```
ListPlot[Take[dB[Abs[Fourier[numGammaPoints-numGammaTheory]]],128],
        PlotJoined->True];
```



Finally, the all-pole gammatone approximation shown in Section 3.5 has the impulse response shown in the plot below.

```
numAllPoleFilt = Simplify[N[10^30*twopolefilt^4/.B->2*Pi*125/.
            cf->1000/.
            T->1/16000]];
numAllPolePoints = FindImpulseResponse[numAllPoleFilt,250];
ListPlot[numAllPolePoints, PlotJoined->True,PlotRange->All];
```



We use the following function to calculate the filter coefficients and put them into Standard C notation. We can then plug these equations directly into our filter design program. (Note, we need the z^{-2} coefficient to be equal to one. Thus we divide both the numerator and the denominator by B0.)

```
FilterForm[filt_] :=
    Block[{term, B0},
        B0 = Coefficient[Denominator[filt],z,2];
        Print["B = 2*Pi*1.019*24.7*(4.37*cf/1000 + 1);"];
        term = Coefficient[Numerator[filt],z,2]/B0;
        Print["A0 = ", CForm[term],";"];
        term = Coefficient[Numerator[filt],z,1]/B0;
        Print["A1 = ", CForm[term],";"];
        term = Coefficient[Numerator[filt],z,0]/B0;
        Print["A2 = ", CForm[term],";"];
        term = Coefficient[Denominator[filt],z,2]/B0;
        Print["B0 = ", CForm[term],";"];
        term = Coefficient[Denominator[filt],z,1]/B0;
        Print["B1 = ", CForm[term],";"];
        term = Coefficient[Denominator[filt],z,0]/B0;
        Print["B2 = ", CForm[term],";"];
    ]
```

```
FilterForm[zfilt1]
```

```
B = 2*Pi*1.019*24.7*(4.37*cf/1000 + 1);
A0 = T;
A1 = -(2*T*Cos(2*cf*Pi*T)/Power(E,B*T) +

      2*Sqrt(3 + Power(2,3/2))*T*Sin(2*cf*Pi*T)/Power(E,B*T))/2;
A2 = 0;
B0 = 1;
B1 = -2*Cos(2*cf*Pi*T)/Power(E,B*T);
B2 = Power(E,-2*B*T);
```

```
FilterForm[zfilt2]
```

```
B = 2*Pi*1.019*24.7*(4.37*cf/1000 + 1);
A0 = T;
A1 = -(2*T*Cos(2*cf*Pi*T)/Power(E,B*T) -

      2*Sqrt(3 + Power(2,3/2))*T*Sin(2*cf*Pi*T)/Power(E,B*T))/2;
A2 = 0;
B0 = 1;
B1 = -2*Cos(2*cf*Pi*T)/Power(E,B*T);
B2 = Power(E,-2*B*T);
```

```
FilterForm[zfilt3]
```

```
B = 2*Pi*1.019*24.7*(4.37*cf/1000 + 1);
A0 = T;
A1 = -(2*T*Cos(2*cf*Pi*T)/Power(E,B*T) +

      2*Sqrt(3 - Power(2,3/2))*T*Sin(2*cf*Pi*T)/Power(E,B*T))/2;
A2 = 0;
B0 = 1;
B1 = -2*Cos(2*cf*Pi*T)/Power(E,B*T);
B2 = Power(E,-2*B*T);
```

```
FilterForm[zfilt4]

B = 2*Pi*1.019*24.7*(4.37*cf/1000 + 1);
A0 = T;
A1 = -(2*T*Cos(2*cf*Pi*T)/Power(E,B*T) -

      2*Sqrt(3 - Power(2,3/2))*T*Sin(2*cf*Pi*T)/Power(E,B*T))/2;
A2 = 0;
B0 = 1;
B1 = -2*Cos(2*cf*Pi*T)/Power(E,B*T);
B2 = Power(E,-2*B*T);
```

■ 4.2 MATLAB Implementation

We can also generate code that will implement the Gammatone filters using the MATLAB program [MathWorks1989]. First we need to figure out what the gain is at the center frequency. By default we normalize each filter's gain so it is one at the center frequency.

```
gammaGain = Simplify[FilterGain[GammaFilt,cf,1/T]]

((-2 E^(4 I cf Pi T) T + 2 E^(-(B T) + 2 I cf Pi T) T
      (Cos[2 cf Pi T] - Sqrt[3 - 2^(3/2)] Sin[2 cf Pi T]))
   (-2 E^(4 I cf Pi T) T + 2 E^(-(B T) + 2 I cf Pi T) T
      (Cos[2 cf Pi T] + Sqrt[3 - 2^(3/2)] Sin[2 cf Pi T]))
   (-2 E^(4 I cf Pi T) T + 2 E^(-(B T) + 2 I cf Pi T) T
      (Cos[2 cf Pi T] - Sqrt[3 + 2^(3/2)] Sin[2 cf Pi T]))
   (-2 E^(4 I cf Pi T) T + 2 E^(-(B T) + 2 I cf Pi T) T
      (Cos[2 cf Pi T] + Sqrt[3 + 2^(3/2)] Sin[2 cf Pi T]))) /
 (-2/E^(2 B T) - 2 E^(4 I cf Pi T) + 2 (1 + E^(4 I cf Pi T))/E^(B T))^4
```

Now for each center frequency we need to figure out the bandwidth at that point. Once we know the bandwidth we can design compute the filter coefficients. This is easy, except that the MATLAB filter function wants the coefficient of the highest power of z in the denominator to equal one. Thus we divide all coefficients by the value of this number before we simplify the coefficients. While we are at it, we also divide the numerator by the gain at the center frequency. This normalizes the gain of each filter so it is unity at its center frequency. (The filter will have non-zero phase at its center frequency.)

We use the *Mathematica* function *FortranForm* because it produces output that is closest to MATLAB's format. But, we will still need to edit the functions. MATLAB assumes that arrays are multiplied as if they were matrices. In this case, we want to operate on these arrays as if they were vectors of individual numbers. Thus element-by-element multiply is indicated by ".*" and

element-by-element division is indicated by "./". In addition, unlike Fortran which uses "**" to indicate exponetiation, MATLAB uses "^".

```
expandedGamma=Expand[Numerator[GammaFilt]]/
                Expand[Denominator[GammaFilt]]

(16*T^4*z^8 - (64*T^4*z^7*Cos[2*cf*Pi*T])/E^(B*T) +
    (96*T^4*z^6*Cos[2*cf*Pi*T]^2)/E^(2*B*T) -
    (64*T^4*z^5*Cos[2*cf*Pi*T]^3)/E^(3*B*T) +
    (16*T^4*z^4*Cos[2*cf*Pi*T]^4)/E^(4*B*T) -
    (96*T^4*z^6*Sin[2*cf*Pi*T]^2)/E^(2*B*T) +
    (192*T^4*z^5*Cos[2*cf*Pi*T]*Sin[2*cf*Pi*T]^2)/E^(3*B*T) -
    (96*T^4*z^4*Cos[2*cf*Pi*T]^2*Sin[2*cf*Pi*T]^2)/E^(4*B*T) +
    (16*T^4*z^4*Sin[2*cf*Pi*T]^4)/E^(4*B*T))/
  (16/E^(8*B*T) + (64*z^2)/E^(6*B*T) + (96*z^4)/E^(4*B*T) +
    (64*z^6)/E^(2*B*T) + 16*z^8 - (128*z*Cos[2*cf*Pi*T])/E^(7*B*T) -
    (384*z^3*Cos[2*cf*Pi*T])/E^(5*B*T) -
    (384*z^5*Cos[2*cf*Pi*T])/E^(3*B*T) -
    (128*z^7*Cos[2*cf*Pi*T])/E^(B*T) +
    (384*z^2*Cos[2*cf*Pi*T]^2)/E^(6*B*T) +
    (768*z^4*Cos[2*cf*Pi*T]^2)/E^(4*B*T) +
    (384*z^6*Cos[2*cf*Pi*T]^2)/E^(2*B*T) -
    (512*z^3*Cos[2*cf*Pi*T]^3)/E^(5*B*T) -
    (512*z^5*Cos[2*cf*Pi*T]^3)/E^(3*B*T) +
    (256*z^4*Cos[2*cf*Pi*T]^4)/E^(4*B*T))
```

```
MatLabFilters[filt_] :=
    Block[{term,B0,gain},
        Print["cf = ", FortranForm[cf/.fixedChannelCF],";"];
        Print["EarQ = ",EarQ/.GlasbergBandwidthParms,";"];
        Print["minBW = ",minBW/.GlasbergBandwidthParms,";"];
        Print["order = ",order/.GlasbergBandwidthParms,";"];
        Print["ERB = ",FortranForm[ERB[cf,EarQ,minBW,order]],";"];
        Print["B = 1.019*2*pi*B;"];
        B0 = Coefficient[Denominator[filt],z,8];
        Print["gain = abs(", FortranForm[gammaGain],");"];
        term = Simplify[Coefficient[Numerator[filt],z,8]/B0];
        Print["forward(:,1) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Numerator[filt],z,7]/B0];
        Print["forward(:,2) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Numerator[filt],z,6]/B0];
        Print["forward(:,3) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Numerator[filt],z,5]/B0];
        Print["forward(:,4) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Numerator[filt],z,4]/B0];
        Print["forward(:,5) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,8]/B0];
        Print["feedback(:,1) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,7]/B0];
        Print["feedback(:,2) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,6]/B0];
        Print["feedback(:,3) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,5]/B0];
        Print["feedback(:,4) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,4]/B0];
        Print["feedback(:,5) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,3]/B0];
        Print["feedback(:,6) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,2]/B0];
        Print["feedback(:,7) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,1]/B0];
        Print["feedback(:,8) = ", FortranForm[term],";"];
        term = Simplify[Coefficient[Denominator[filt],z,0]/B0];
        Print["feedback(:,9) = ", FortranForm[term],";"];
        ]
```

MatLabFilters[expandedGamma]

```
cf = -(EarQ*minBW) + E**

     (i*(-Log(fn + EarQ*minBW) + Log(lowFreq + EarQ*minBW))/numChanne

    (fn + EarQ*minBW);
EarQ = 9.26449;
minBW = 24.7;
order = 1;
ERB = ((cf/EarQ)**order + minBW**order)**(1/order);
B = 1.019*2*pi*B;
gain = abs((-2*E**((0,4)*cf*Pi*T)*T +

      2*E**(-(B*T) + (0,2)*cf*Pi*T)*T*

       (Cos(2*cf*Pi*T) - Sqrt(3 - 2**(3/2))*Sin(2*cf*Pi*T)))*

    (-2*E**((0,4)*cf*Pi*T)*T +

      2*E**(-(B*T) + (0,2)*cf*Pi*T)*T*

       (Cos(2*cf*Pi*T) + Sqrt(3 - 2**(3/2))*Sin(2*cf*Pi*T)))*

    (-2*E**((0,4)*cf*Pi*T)*T +

      2*E**(-(B*T) + (0,2)*cf*Pi*T)*T*

       (Cos(2*cf*Pi*T) - Sqrt(3 + 2**(3/2))*Sin(2*cf*Pi*T)))*

    (-2*E**((0,4)*cf*Pi*T)*T +

      2*E**(-(B*T) + (0,2)*cf*Pi*T)*T*

       (Cos(2*cf*Pi*T) + Sqrt(3 + 2**(3/2))*Sin(2*cf*Pi*T)))/

   (-2/E**(2*B*T) - 2*E**((0,4)*cf*Pi*T) +

      2*(1 + E**((0,4)*cf*Pi*T))/E**(B*T))**4);
forward(:,1) = T**4;
forward(:,2) = -4*T**4*Cos(2*cf*Pi*T)/E**(B*T);
forward(:,3) = 6*T**4*Cos(4*cf*Pi*T)/E**(2*B*T);
forward(:,4) = -4*T**4*Cos(6*cf*Pi*T)/E**(3*B*T);
forward(:,5) = T**4*Cos(8*cf*Pi*T)/E**(4*B*T);
feedback(:,1) = 1;
feedback(:,2) = -8*Cos(2*cf*Pi*T)/E**(B*T);
feedback(:,3) = 4*(4 + 3*Cos(4*cf*Pi*T))/E**(2*B*T);
feedback(:,4) = -8*(6*Cos(2*cf*Pi*T) + Cos(6*cf*Pi*T))/E**(3*B*T);
feedback(:,5) = 2*(18 + 16*Cos(4*cf*Pi*T) + Cos(8*cf*Pi*T))/E**(4*B*T
feedback(:,6) = -8*(6*Cos(2*cf*Pi*T) + Cos(6*cf*Pi*T))/E**(5*B*T);
feedback(:,7) = 4*(4 + 3*Cos(4*cf*Pi*T))/E**(6*B*T);
feedback(:,8) = -8*Cos(2*cf*Pi*T)/E**(7*B*T);
feedback(:,9) = E**(-8*B*T);
```

Here is the MATLAB code to design this filter bank.

```
function [forward, feedback]=MakeERBFilters(fs,numChannels,lowFreq)
% [forward, feedback]=MakeERBFilters(fs,numChannels) computes the
% filter coefficients for a bank of Gammatone filters.  These
% filters were defined by Patterson and Holdworth for simulating
% the cochlea.  The results are returned as arrays of filter
% coefficients.  Each row of the filter arrays (forward and feedback)
% can be passed to the MatLab "filter" function, or you can do all
% the filtering at once with the ERBFilterBank() function.
%
% The filter bank contains "numChannels" channels that extend from
% half the sampling rate (fs) to "lowFreq".

T=1/fs;
% Change the following parameters if you wish to use a different
% ERB scale.
EarQ = 9.26449;              %  Glasberg and Moore Parameters
minBW = 24.7;
order = 1;

% All of the following expressions are derived in Apple TR #35, "An
% Efficient Implementation of the Patterson-Holdsworth Cochlear
% Filter Bank."
cf = -(EarQ*minBW)+exp((1:numChannels)'*(-log(fs/2 + EarQ*minBW) + ...
                         log(lowFreq + EarQ*minBW))/numChannels) ...
                          *(fs/2 + EarQ*minBW);
ERB = ((cf/EarQ).^order + minBW^order).^(1/order);
B=1.019*2*pi*ERB;
gain = abs((-2*exp(4*i*cf*pi*T)*T + ...
             2*exp(-(B*T) + 2*i*cf*pi*T).*T.* ...
              (cos(2*cf*pi*T) - sqrt(3 - 2^(3/2))* ...
               sin(2*cf*pi*T))) .* ...
           (-2*exp(4*i*cf*pi*T)*T + ...
             2*exp(-(B*T) + 2*i*cf*pi*T).*T.* ...
              (cos(2*cf*pi*T) + sqrt(3 - 2^(3/2)) * ...
               sin(2*cf*pi*T))).* ...
           (-2*exp(4*i*cf*pi*T)*T + ...
             2*exp(-(B*T) + 2*i*cf*pi*T).*T.* ...
              (cos(2*cf*pi*T) - ...
               sqrt(3 + 2^(3/2))*sin(2*cf*pi*T))) .* ...
           (-2*exp(4*i*cf*pi*T)*T+2*exp(-(B*T) + 2*i*cf*pi*T).*T.* ...
           (cos(2*cf*pi*T) + sqrt(3 + 2^(3/2))*sin(2*cf*pi*T))) ./ ...
          (-2 ./ exp(2*B*T) - 2*exp(4*i*cf*pi*T) +  ...
           2*(1 + exp(4*i*cf*pi*T))./exp(B*T)).^4);
feedback=zeros(length(cf),9);
forward=zeros(length(cf),5);
forward(:,1) = T^4 ./ gain;
forward(:,2) = -4*T^4*cos(2*cf*pi*T)./exp(B*T)./gain;
forward(:,3) = 6*T^4*cos(4*cf*pi*T)./exp(2*B*T)./gain;
forward(:,4) = -4*T^4*cos(6*cf*pi*T)./exp(3*B*T)./gain;
forward(:,5) = T^4*cos(8*cf*pi*T)./exp(4*B*T)./gain;
feedback(:,1) = ones(length(cf),1);
```

```
feedback(:,2) = -8*cos(2*cf*pi*T)./exp(B*T);
feedback(:,3) = 4*(4 + 3*cos(4*cf*pi*T))./exp(2*B*T);
feedback(:,4) = -8*(6*cos(2*cf*pi*T) + cos(6*cf*pi*T))./exp(3*B*T);
feedback(:,5) = 2*(18 + 16*cos(4*cf*pi*T) +
cos(8*cf*pi*T))./exp(4*B*T);
feedback(:,6) = -8*(6*cos(2*cf*pi*T) + cos(6*cf*pi*T))./exp(5*B*T);
feedback(:,7) = 4*(4 + 3*cos(4*cf*pi*T))./exp(6*B*T);
feedback(:,8) = -8*cos(2*cf*pi*T)./exp(7*B*T);
feedback(:,9) = exp(-8*B*T);
```

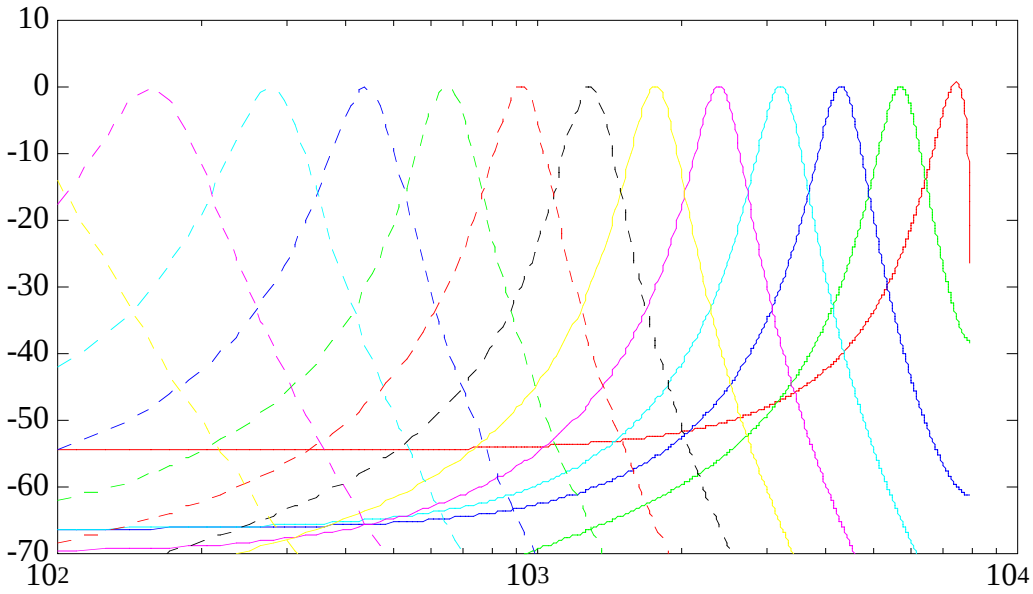
The following MATLAB function applies the bank of filters to a signal, producing an array of output signals.

```
function y=ERBFilterBank(forward,feedback,x)
% y=ERBFilterBank(forward,feedback,x)
% This function filters the waveform x with the array of filters
% specified by the forward and feedback parameters.  Each row
% of the forward and feedback parameters are the parameters
% to the Matlab builtin function "filter".

[rows, cols]=size(feedback);
y=zeros(rows,length(x));
for i=1:rows
 y(i,:)=filter(forward(i,:),feedback(i,:),x);
end
```

Finally this code is tested by entering the following MATLAB (Version 3.5) commands.

```
impulse=[1 zeros(1,1023)];
[ERBforward,ERBfeedback]=MakeERBFilters(16000,64,20);
y=ERBFilterBank(ERBforward,ERBfeedback,impulse);

response=20*log10(abs(fft(y(1:5:64,:)')));
freqScale=(0:1023)/1024*16000;
axis([2 4 -70 10])
semilogx(freqScale(1:512),response(1:512,:))
```

(Note, in Version 4 of MATLAB, the axis are specified in the *axis* command as [20 10000 -70 10].) This results in the following plot for the frequency response of every fifth channel.



A similar procedure can be used to design the all-pole implementation but this is not shown in this report.

5.0 Acknowledgements

This approach (Laplace transform of the Gammatone function) was suggested to the author by Richard F. Lyon. Much of the credit for this report should go to Roy Patterson and John Holdsworth, who developed the original model. We also appreciate the help we have received from Roy Patterson and Peter Assmann in writing this report.

6.0 References

[Cooke1991] Martin Cooke, "Modelling Auditory Processing and Organisation," PhD Dissertation, University of Sheffield, Computer Science Department, May 1991.

[CRC1966] Chemical Rubber Company, *Handbook of Chemistry and Physics*, Robert C. Weast and Samuel M. Selby, eds., The Chemical Rubber Company, 1966.

[Glasberg1990] B. R. Glasberg and B. C. J. Moore, "Derivation of auditory filter shapes from notched-noise data." *Hearing Research*, vol. 47, 1990, pp. 103-108.

[Greenwood1990] Donald D. Greenwood, "A cochlear frequency-position function for several species—29 years later," *Journal of the Acoustical Society of America*, Vol. 87 (6), June 1990, pp. 2592-2605.

[MathWorks1989] The MathWorks, Inc., *MATLAB™ for Macintosh™ Computers*, South Natick, MA, 1989.

[Patterson1992] R. D. Patterson, K. Robinson, J. Holdsworth, D. McKeown, C.Zhang, and M. H. Allerhand, "Complex sounds and auditory images," In *Auditory Physiology and Perception*, (Eds.) Y Cazals, L. Demany, K.Horner, Pergamon, Oxford, 1992, pp. 429-446.

[Slaney1988] Malcolm Slaney, "Lyon's Cochlear Model," Apple Technical Report #13, Apple Computer Corporate Library, Cupertino, CA 95014, 1988.

[Slaney1993] Malcolm Slaney, "*Mathematica* Filter Design," Apple Technical Report #34, Apple Computer Library, Cupertino, CA 95014, 1993.

[Wolfram1988] Stephen Wolfram, *Mathematica: A System for Doing Mathematics by Computer*, Addison Wesley, Redwood City, CA 1988.