DSP generation of Pink (1/f) Noise

Looking at how to generate pink noise by two methods:

1 - A "pinking" filter for white noise.
2 - The Voss-McCartney algorithm of adding multiple white noise sources at lower and lower octaves. (2006-03-27 Please also see Larry Trammell's "Stochastic Voss-McCartney algorithm".)

Keywords: Pink noise, 1/f noise, 1fnoise, flicker noise, random number generation, DSP.

1999 October 18

Minor updates:

2003 July 10: "pinkish" opcode in Csound.
2005 September 19: new page on Park-Miller-Carta pseudo-random number generators ../rand31/
2006 March 27: Link to Larry Trammell's (RidgeRat's) Stochastic Voss-McCartney algorithm.
2007 January 22: Updated the link to Larry's material.
2010 March 12: Added link to dsprelated.com.
2011 March 20: Added link to Henning Thielemann's paper

Robin Whittle rw@firstpr.com.au

Most of this material is written by other people, especially Allan Herriman, James McCartney, Phil Burk and Paul Kellet - all from the music-dsp mailing list.
Email addresses here have "xxxx." added to confound SPAM robots.

Back to the DSP directory
Back to the main First Principles site, including material on Csound.

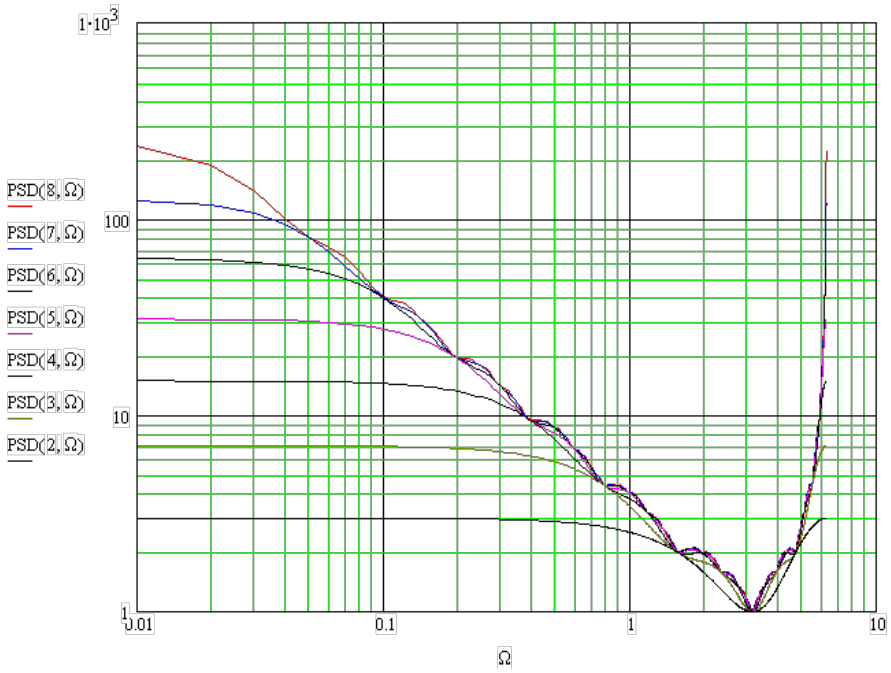
Good news for Csounders

Standard Csound (after version 4.07) has a "pinkish" opcode which generates pink noise or filters external white noise to make it pink. This was written by Phil Burk and John ffitch, and is documented here:

http://kevindumpscore.com/docs/csound-manual/pinkish.html

This was added in May 2000, but I only realised it in 2003 July 10. More information on the Csound music synthesis language is at http://www.csounds.com .

Index



A graph from Allan Herriman's illustrated treatise on the summing of noise sources: allan-2/spectrum2.html .

Introduction

A stream of random numbers constitutes "white" noise - if listened to as an audio signal. The "white" refers to the even distribution of wavelengths in white light, with a particular meaning in the audio or DSP sense: that the power of the noise is distributed evenly over all frequencies, between 0 and some maximum frequency which is typically half the sampling rate. For instance, white noise at a sampling rate of 44,100 Hz will have as much power between 100 and 600 Hz as between 20,000 and 20,500 Hz. To our ears, this seems very bright and harsh.

A 1996 treatise by Joseph S. Wisniewski on the "Colors of Noise", including white, pink, orange, green . . . is at: http://www.msaxon.com/colors.htm . (Also at this site, Martin Saxon's description of the various weighting schemes for measuring noise: http://www.msaxon.com/noise.htm .)

In the natural world, there are many physical processes which produce noise with what is known as a "pink" distribution of power. "Pink" noise has an even distribution of power if the frequency is mapped in a logarithmic scale. A straightforward example would be that there is as much noise power in the octave 200 to 400 Hz as there is in the octave 2,000 to 4,000 Hz. Consequently, it seems, our ears tell us that this is a "natural" even noise.

Statistical genetics Prof. Wentian Li maintains a formidable bibliography on 1/f noise at:

http://linkage.rockefeller.edu/wli/1fnoise/

However, prior to me creating this page, it mentioned nothing to do with generating 1/f noise with Digital Signal Processing techniques.

The purpose of this page is to collect information on the generation of noise - and especially pink noise - digitally. My primary use for DSP pink noise is in software music synthesis, both as an audio signal and as a "control signal", which could have frequencies as low as 0.001 Hz. Hopefully this material will be of value to other fields as well.

(For general DSP information, see the FAQ of Usenet newsgroup comp.dsp, and the FAQs it mentions: http://www.bdti.com/faq/ .)

Pseudo random numbers and white noise

In 1999, when I wrote most of this page, I used a Park Miller PRNG, as I wrote at: ../../csound/ . The one I used was a 31 bit Park-Miller linear congruential generator by Ray Gardner: http://c.snippets.org/snip_lister.php?fname=rg_rand.c .

2005 September 19 update: See my new page on Park-Miller-Carta pseudo-random number generators **../rand31/**

A stream of uncorrelated random numbers constitutes white noise. For instance a uniform noise source with a -10 to +10 range will be made of random numbers with no correlation whatsoever between one and the next, and where there is a 5% chance of each sample being, for instance, in the -10 to -9 range, the 2 to 3 range or any other range which is a 20th of the total peak-to-peak range

The characteristics of pink noise

For the purposes of this discussion, "power" means the average power or energy contained in a signal over a long period of time.

White noise has the same distribution of power for all frequencies, so there is the same amount of power between 0 and 500 Hz, 500 and 1,000 Hz or 20,000 and 20,500 Hz.

Pink noise has the same distribution of power for each octave, so the power between 0.5 Hz and 1 Hz is the same as between 5,000 Hz and 10,000 Hz.

Since power is proportional to amplitude squared, the energy per Hz will decline at higher frequencies at the rate of about −3dB per octave. To be *absolutely* precise, the rolloff should be −10dB/decade, which is about 3.0102999 dB/octave.

The uses of pink noise

The most obvious use of pink noise is as an audio signal, to be used directly, to be filtered or to be used to modulate something.

I am also interested in pink noise way below 1 Hz as a control signal for simulating randomly fluctuating aspects of music. For instance, I might want some aspect of a piece to fluctuate on a minute-by-minute basis, so I need random numbers with energy at 0.01 Hz and below.

My particular interest is being able to sculpt such control signals from a pink noise source purely by the use of filters. The idea would be that I could have a band-pass filter with a certain bandwidth in octaves (or fractions of an octave) and that I could choose to set its frequency as I liked, without affecting the RMS level of its output. Without a pink noise source – for instance by using a white noise source – it becomes very difficult to adjust the piece by changing the frequency of the sculpting filter, because this also affects the resultant signal level,

Ideally, for control purposes (eg. "k" rate noise in Csound) I would like to be able to specify noise with:

1 – A certain lower bounding frequency. Eg. 0.1 Hz. Below that, there would either be little energy, or the energy would remain flat per Hz, rather than rising per Hz to give the 3dB/octave characteristic of pink noise. So it would be "white" below 0.1 Hz.

2 – A certain "RMS level per octave. For instance 5.0 RMS per octave. Therefore a perfect filter which excluded everything

but an octave - no matter which octave above the lower bounding frequency - would average an RMS level of 5.0. Noise being noise, it would take a long time to average out the fluctuations to measure this accurately.

3 - It might also be desirable to specify an upper bounding frequency, to reduce computational load where high frequencies were not required.

In the future I intend to write a Csound / Quasimodo unit generator for "a" or "k" rate output, of an "industrial" quality, rather than "analytical" grade. Industrial or technical grade nitric acid specifies it being of a particular minimum and approximate strength. Analytical grade specifies exactly its strength and the tolerance for that specification, as well as noting the maximum permissible levels of the most important contaminants. Such parameters would be "i" rate: set at the start of the ugen's instantiate, not changeable over time. So specifying a lower limit frequency limit of the pink quality of the noise would result in that boundary being set to the nearest octave, not precisely.

It is possible to conceive of a pink noise generator with "k" rate control of level, upper and lower boundary frequencies and the slopes of those boundaries. In this way, precise control over the noise frequency distribution could be achieved without changing the RMS level of the noise. I will leave this idea for now, but it would be mighty handy!

Filtering white noise to make it pink

The simplest DSP filters are -6dB/octave. However, a DSP or analogue electronic low pass filter with a -3dB/octave response is (or rather, *was*) a rare beast indeed. Here are three filters which do the job. (Paul Kellet contributed two earlier filters to those listed here.)

Such a filter would be fed with white noise to produce pink, within certain limits of accuracy.

Below the algorithms is Allan Herriman's graphical analysis of the response of these three filters.

The first description I am aware of is from Robert Bristow-Johnson <pbjrbj@xxxx.viconet.com> posting to the Music-DSP list on 30 June 1998:

> (This is **rbj = (Red)** Robert Bristow-Johnson's three pole and three zero filter.)
>
> ```
> >  "Orfanidis also mentions a clever way to get reasonably good 1/f
> >  noise: sum together n randh's, where each randh is running an
> >  octave slower than the preceding (one):"
> >
> ```

> This is a reference to Sophocles Orfanidis' book "Introduction to
> Signal Processing":
>
> http://www.prenhall.com/books/esm_0132091720.html
>
> This sounds like a pretty good way to do it.

another method that Orfanidis mentions came from a comp.dsp post of mine.
it's just a simple "pinking" filter to be applied to white noise. since
the rolloff is -3 dB/octave, -6 dB/octave (1st order pole) is too steep and
0 dB/octave is too shallow.

an equiripple approximation to the ideal pinking filter can be realized by
alternating real poles with real zeros. a simple 3rd order solution that i
obtained is:

pole	zero
0.99572754	0.98443604
0.94790649	0.83392334
0.53567505	0.07568359

the response follows the ideal -3 dB/octave curve to within + or - 0.3 dB
over a 10 octave range from 0.0009*nyquist to 0.9*nyquist. probably if i
were to do it over again, i'd make it 5 poles and 4 zeros.

r b-j
pbjrbj@xxxx.viconet.com a.k.a. robert@xxxx.audioheads.com
a.k.a. robert@xxxx.wavemechanics.com

"Don't give in to the Dark Side. Boycott intel and microsoft."

From the Music-DSP code archive: http://music.columbia.edu/cmc/music-dsp/ (Was at http://shoko.calarts.edu/~glmrboy/musicdsp/dspsource.html) Paul Kellet <paul.kellett@xxxx.maxim.abel.co.uk> http://www.abel.co.uk/~maxim/ has been updated on several occasions and now (1999 October 17) contains two implementations, located here: http://shoko.calarts.edu/~glmrboy/musicdsp/sourcecode/pink.txt which is "is accurate to within +/-0.05dB above 9.2Hz (44100Hz sampling rate)."

2011-03-20 update: the music-DSP archives are at: http://www.musicdsp.org/showmany.php contains several items relating to pink noise, but I am not sure which of these, if any, are the pink.txt referred to above.

On 17 October 1999, Paul put up a further refinement: "instrumentation grade" and "economy" filters.

This is an approximation to a -10dB/decade filter using a weighted sum
of first order filters. It is accurate to within +/-0.05dB above 9.2Hz
(44100Hz sampling rate). Unity gain is at Nyquist, but can be adjusted
by scaling the numbers at the end of each line.

(This is **pk3 = (Black)** Paul Kellet's refined method in Allan's analysis.)

```
b0 = 0.99886 * b0 + white * 0.0555179;
b1 = 0.99332 * b1 + white * 0.0750759;
b2 = 0.96900 * b2 + white * 0.1538520;
b3 = 0.86650 * b3 + white * 0.3104856;
b4 = 0.55000 * b4 + white * 0.5329522;
b5 = -0.7616 * b5 - white * 0.0168980;
pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
b6 = white * 0.115926;
```

An 'economy' version with accuracy of +/-0.5dB is also available.

(This is **pke = (Blue)** Paul Kellet's economy method.)

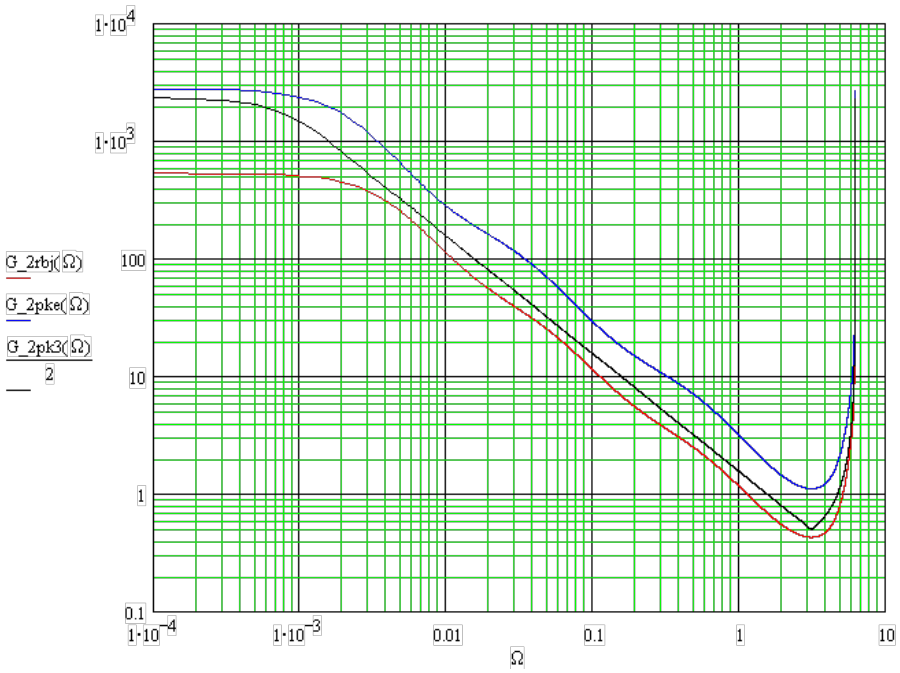
```
b0 = 0.99765 * b0 + white * 0.0990460;
b1 = 0.96300 * b1 + white * 0.2965164;
b2 = 0.57000 * b2 + white * 1.0526913;
tmp = b0 + b1 + b2 + white * 0.1848;
```

Here is Allan Herriman's graphical analysis of the response of these three filters.

Magnitude Squared Response of Three Pinking Filters

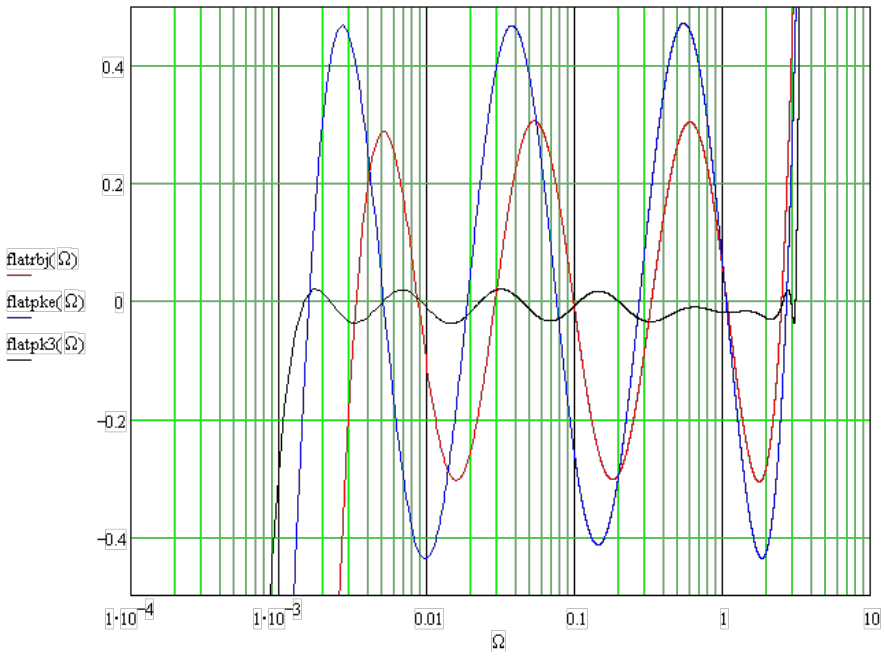
Allan Herriman 18 October 1999

rbj = (Red) Robert Bristow-Johnson's three pole and three zero filter
pk3 = (Black) = Paul Kellet's refined method
pke = (Blue) Paul Kellet's economy method



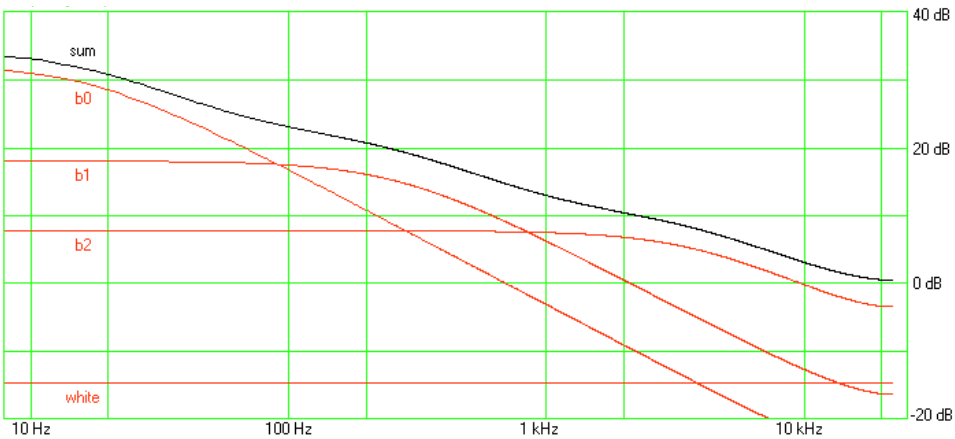
Flatness of Three Pinking Filters

Vertical scale is 0.2dB/division.



I asked Paul about the algorithm he used to devise these highly tweaked filters. He replied with the following text and image.

> It comes back to the roll-off of a first order low-pass filter being too steep at -6dB/octave. The only thing I could think of with a softer roll-off was the transition from 0dB/oct to -6dB/octave at the "knee" of such a filter. By positioning enough of these knees in a "staircase" a good -3dB/oct slope can be made.
>
> A slight rise in level occurs near Nyquist but this can be counteracted by combining a little unfiltered, high-pass filtered, and/or delayed signal with the output. Each version of the filter was designed by hand using a simple Visual Basic program to plot the impulse response (with a +3dB/oct emphasis) as the coefficients were adjusted.
>
> The attached picture shows how the "economy" version is constructed.



I am glad to see that these highly-tweaked filters are carefully hand-crafted!

As part of a discussion on flattening the Voss algorithm's frequency response, Allan Herriman contributed the following on flattening the response of the filtered white noise approach. (This was before Paul Kellet's "instrumentation" and "economy" pinking filters of 17 October 1999.)

```
This trick could also be used to reduce the ripple of the filtered white
noise method.  However in this case I think it is much more efficient (and
easier) to just add more poles and zeros to the filter.

Filters will have all the poles and zeros real (i.e. 1st order sections).
The poles and zeros will alternate, and there will be a constant ratio
between the frequencies.  This ratio determines the ripple.
E.g. if the ratio is 2 and we start with a pole at 1Hz, we get the following:

Pole    Zero
1Hz     2Hz
4Hz     8Hz
16Hz    32Hz
...

Doing some simulations in PSpice (!) gave me the following approximate
ripple values:

pole/zero
Ratio           Ripple
1.414           ~0dB
2               0.03dB
2.5             0.16dB
3               0.4dB
3.5             0.7dB
4               1dB

If there is only a small number of poles and zeros (e.g. rb-j's posting had
three of each) then the optimal result will not have an exact logarithmic
spacing of frequencies.

I guess it is possible to use some optimisation method to get the best pole
and zero locations for a given amount of ripple, but I haven't tried this.
The pole and zero locations might also need to be warped to take sampling
effects into account.

I suggest having one more pole than zero, so that the noise above the range
of interest is brown, and the noise below the range of interest is white.
This would probably not matter though.
```

The Voss algorithm

On 30 June 1998, Thomas Hudson sent me this C++ code which implements the Voss algorithm, which creates pink noise by adding a series of white noise sources at successively lower octaves:

```
In case you have trouble finding the M. Gardner article, I have a C++
class that implements Voss's algorithm:

#include <iostream>
#include <stdlib.h>

class PinkNumber
{
private:
  int max_key;
  int key;
  unsigned int white_values[5];
  unsigned int range;
public:
  PinkNumber(unsigned int range = 128)
    {
      max_key = 0x1f; // Five bits set
      this->range = range;
      key = 0;
      for (int i = 0; i < 5; i++)
```

```
 white_values[i] = rand() % (range/5);
    }
  int GetNextValue()
    {
      int last_key = key;
      unsigned int sum;

      key++;
      if (key > max_key)
 key = 0;
      // Exclusive-Or previous value with current value. This gives
      // a list of bits that have changed.
      int diff = last_key ^ key;
      sum = 0;
      for (int i = 0; i < 5; i++)
 {
   // If bit changed get new random number for corresponding
   // white_value
   if (diff & (1 << i))
     white_values[i] = rand() % (range/5);
   sum += white_values[i];
 }
      return sum;
    }
};
#ifdef DEBUG
main()
{
  PinkNumber pn;

  for (int i = 0; i < 100; i++)
    {
      cout << pn.GetNextValue() << endl;
    }
}
#endif // DEBUG
```

I replied:

```
Thanks for this code.  It confirms my understanding of straight
"sample and hold" of lower octave white noise sources, and even
weighting of them all.

The only thing is that the fastest changing noise source in this
code changes every second sample.  Shouldn't there also be white
noise on each sample?  In which case, it would be best make the
random values divided by 6 rather than 5 and set "sum" to "rans() %
(range/6) rather than 0.
```

Thomas replied:

```
You're right. Actually I may have modified this slightly from
the original article. Or it may be a mistake. I think I was
experimenting w/ various ranges and numbers of white values.
When I did this I was using the resulting pink number as a
lookup value in a table of notes. I even may have a version
somewhere that had a "granularity" parameter, allowing one
to not only to specify the range, but the number of white
values. I have been meaning to plot the affect of different
numbers of white values...
```

The Voss-McCartney algorithm

In late August and early September, the 1/f noise debate reared its head in a most constructive form on the Music-DSP mailing list: http://shoko.calarts.edu/~glmrboy/musicdsp/music-dsp.html .

Based on that discussion and feedback and contributions from the

participants, here is the key elements of the Voss McCartney algorithm and its implementation.

James McCartney <asynth@xxxx.io.com> 2 Sep 1999 21:00:30 -0600

```
At 3:52 PM -0600 9/2/99, Stephan M. Sprenger wrote:
> > I've heard of a proposal for making pink noise by addition of multiple
> > noise sources.  (I haven't tried it or analysed it myself, so I can't
> > say if it's any good.)
>
>
> Allan,
>
> this is the algorithm I referred to as Voss' method in my previous
> posting. It's described in M. Gardner, "White and Brown Music, Fractal
> Curves and One-Over-f Fluctuations", Sci.Amer., 16 (1978) p.288
> according to Orfanidis who mentions it in "Introduction to Signal
> Processing". He also gives an implementation of it, but I haven't tried
> it yet so I cannot really comment.


I posted an improvement to this algorithm a while back.
Here it is again:


Here's how to improve the Gardner pink noise generator. Gardner adds up
several uniform random number generators that are evaluated in octave time
intervals. The pattern is as follows:


x x x x x x x x x x x x x x x x
x   x   x   x   x   x   x   x   x
x       x       x       x       x
x               x               x
x                               x


Now the problem with the above is that on some samples you have to
add up a lot more random values than others. This can also cause
large discontinuities in the wave when lots of the values
change at once.


By rearranging the order that the random generators change the
load can be made even, the operation can be made more efficient
and the deviation from any one sample to another is bounded by a
constant value every sample. Here's the pattern, a tree structure:


x x x x x x x x x x x x x x x x
 x   x   x   x   x   x   x   x
   x       x       x       x
       x               x
               x


Only one of the generators changes each sample. This makes the
load constant. It also means that you can update the sum by
subtracting the previous value of a generator and adding
the new value, instead of summing them all together.


You can determine which random number needs to be changed each
sample by incrementing a counter and counting the trailing zeroes
in the word. The number of trailing zeroes is the index of
the random number to change. Number of trailing zeroes can
be determined very quickly on a machine with a count leading
zeroes instruction.


Remainder of excercise left to the reader.
```

The patterns above show time moving horizontally to the right, and each line being a random number, or rather a square-wave sampled white noise source.

Later, James clarified this by noting that ever sample also contained a white noise sample:

```
> The top end of the spectrum wasn't as good.  The cascade of sin(x)/x
> shapes that I predicted in my other post was quite obvious.
> Ripple was only about 2dB up to Fs/8 and 4dB up to Fs/5.  The response
> was about 5dB down at Fs/4 (one of the sin(x)/x nulls), and there was
```

> a deep null at Fs/2.
> (These figures are a bit rough. More averaging would have helped.)

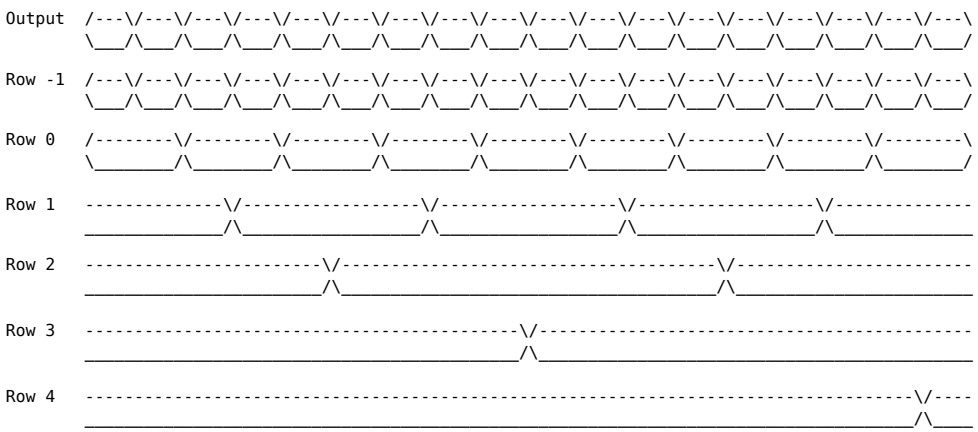
You can improve the top octave somewhat by adding a white noise generator at the same amplitude as the others. Which fills in the diagram as follows:

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x x x x x x x x x x x x x x x x
 x   x   x   x   x   x   x   x
   x       x       x       x
       x               x
               x
```

It'll still be bumpy up there, but the nulls won't be as deep.

Here is my ASCII diagram of the waves which I think James' last diagram represants:

Output samples are on the top row, and are the sum of all the other rows at that time.



"Row -1" is the white noise added to each output sample. Rows 0 to 4 are the rows of sampled white noise which are all mixed together.

Allan posted an ASCII graphic explanation of the frequency spectrums of the various noise sources and how they all add togther. See allan-2/spectrum2.html for a more precise theoretical and graphical analysis.

Allan Herriman Sat, 04 Sep 1999 05:18:24 +1000

The statistics of the random numbers are identical. However, each row represents a sequence being output at a different rate. Another way of looking at this is that there is a true white noise source (infinite bandwidth), with a different sample and hold for each row. The sample and hold (or zero order hold) has a sin(x)/x frequency response.

(I think we should be thinking in terms of power, so the response is ((sin(x)/x)^2)).

Since each row samples at a different rate, its frequency response will be different.

(Sorry about the ASCII art.)

Top Row PSD:
((sin(x)/x)^2 shape)

```
^
|* *  *
|           *
|                *
|                   *    *
```

```
|                  * *  *   *
+---------------------*------*-----*-->f

Second Top Row PSD:
(shape scrunched by factor of 2 on frequency axis.)
^
|* *
|    *
|      *
|       *   *
|        * *  *   *
+----------*----*-----*----------------->f

Bottom Row PSD:
(shape scrunched by factor of 2^N on frequency axis.)
^
|*
|*
|*
|*
|**
+*-*------------------------------------->f

Sum of all PSDs:
(nice -10dB/decade slope up to Fs/2)
^
|* * *
|    * * *
|        * * *
|             * * *
|                  * * * * *
+---------------------------------------->f

Trust me, this works.  I just spent a whole evening testing it.
Yes, I know.  I need to get a life :(
```

Phil Burk posted some code, which was apparently similar to what Allan was going to post. This is an integer implentation, and rather than addiing all the rows for each sample, when a row changes, it subtracts the row's previous value from the running total and adds the new row value (a random number). There is a white noise signal added to this running total to produce the output value - as I suggested above, the equivalent of a "Row -1" which changes each sample.

The final version of Phil Burk's code is here:

phil_burk_19990905_patest_pink.c
phil_burk_19990905_patest_pink.c .

See also the "inner loop" code below from James McCartney.

Following that there was quite a bit of discussion about how to code the selection of which row to update. Phil's code uses a shift and test loop to count the trailing number of zeroes, which then selects the row to update. The Power PC CPU has an instruction for this, and so too apparently does the Pentium.

All this is in the list archive in early September 1999 and I won't reproduce it here.

There was also discussion of when no row (rows 0 to 4 in my

diagram above) would change, on infrequent occasions. James McCartney responded that the occasional unchanged sample is not much of a problem. I agree, especially since we are adding in white noise ("row −1" in my diagram) on every sample. He also commented on coding strategies to optimise speed when looking for the 0 state of a counter in a loop. I haven't really followed this.

James McCartney Sun, 5 Sep 1999 16:18:52 −0600

```
At 12:34 PM -0600 9/5/99, Phil Burk wrote:
> James McCartney wrote:
> >
> > At 10:39 AM -0600 9/4/99, Phil Burk wrote:
> > > Just to test my understanding, I have put together a 'C'
> > > implementation of the Pink Noise algorithm we've been discussing.
> > > Something that wasn't obvious to me at first was that when the
> > > is zero I do not change any any rows.
> >
> > One "row" should change every sample.
>
> I'm not sure I agree with this. The tree diagram that you showed had an
> odd number of columns, 2**N-1. I think the missing column represents a
> sample when no rows change. Otherwise one of the rows changes once too
> often and not at an octave frequency.

It doesn't really matter. If you have enough octaves then the
counter will be zero only infrequently.

Note that with this method the number of octaves you use has
no effect on the computational cost.

If you are using 16 octaves then you get a bit more energy way down at
0.67 Hz @ Fs=44100. If you are using 8 octaves then it will be at 172Hz,
but it is pretty negligible.

It is better to just let it happen than to check for a zero
every time through the loop.



Which brings up another topic.
In a lot of DSP code I see people doing this:

nsmps = blah;
counter = blah;
for (i=0; i<nsmps; ++i) {
        ... stuff ...
        if (--counter == 0) { ..do counter stuff.. }
}



That is a very expensive way to do it, because you check for the
counter everytime, even though most times through the loop the
test fails.

Since you know the value of the counter,  you can predict when
it will run out. Therefore it is much better to do the
following:

nsmps = blah;
counter = blah;
remain = nsmps;
while (remain) {
        jsmps = min(remain, counter);
        for (i=0; i<jsmps; ++i) {
                ... stuff ...
        }
        remain -= jsmps;
        counter -= jsmps;
        if (counter == 0) { ..do counter stuff.. }
}

This eliminates all the unecessary conditional tests thereby
speeding up the inner loop.
```

James McCartney posted his **inner loop** code:

James McCartney Mon, 6 Sep 1999 12:51:50 -0600

```
At 10:14 AM -0600 9/6/99, Phil Burk wrote:
> James McCartney wrote:
> >
> > At 6:22 PM -0600 9/5/99, Phil Burk wrote:
> > > James McCartney wrote:
> > >
> > > > Sadly, since I am not writing pure Macintosh code I can't rely on
> > > > the snazzy PPC instruction to count leading zeros.  So I am using an
> > > > inefficient high level loop to count trailing zeros. This loop fails
> > > > when the counter is zero so I have to check for zero anyway.
> > >
> > > OR the counter with 0xFFFF0000. Then you will have 16 octaves
> > > and your loop will never fail.
> >
> Maybe my tea hasn't kicked in cuz I still don't get it. Suppose I have
> 16 octaves, indexed 0-15. ORing with 0xFFFF0000 means I will sometimes
> get 16 trailing zeros. So wouldn't I still have to check for that case
> to prevent over-indexing my array. I might as well check for zero before
> entering the loop.

Then OR with 0xFFFF8000

> I'm inclined to stick with my original code since it is mathematically
> OK, portable, and acceptably fast. The (cntr==0) branch will only be
> taken once every 2**N times. The other times it only costs a one
> instruction branch not taken because the test follows the generation of
> the index and the condition codes are already set.

Really I think that unless you use an instruction to get the trailing
zeroes, that there is no advantage to my algorithm.
The whole point of it is that with by taking advantage of such an
instruction you get a big speed up over filtering noise, or the
normal Voss/Gardner algorithm.
I think that most processors have such instructions.
I see no reason you can't #ifdef it into your code.

Here is my own inner loop implementation:

unsigned long *dice, counter, prevrand, newrand, seed, total, ifval, k;
float *out;

...

for (i=0; i<nsmps; ++i) {
    k = CTZ(counter);         // count trailing zeroes
    k = k & 15;

    // get previous value of this octave
    prevrand = dice[k];

    // generate a new random value
    seed = 1664525 * seed + 1013904223;
    // shift to move into mantissa bitfield divided by 16
    newrand = seed >> 13;

    // store new value
    dice[k] = newrand;

    // update total
    total += (newrand - prevrand);

    // generate a new random value for the top octave
    seed = 1664525 * seed + 1013904223;
    // shift to move into mantissa bitfield divided by 16
    newrand = seed >> 13;

    // there is a theoretical chance that total + newrand could overflow
    // the mantissa bitfield, but it is *extremely* unlikely due to total
    // being a a gaussian distributed value.
```

```
        // convert to floating point value from 2.0 to 4.0 by masking
        ifval = (total + newrand) | 0x40000000;

        // subtract 3 to get in range from -1 to +1 and output
        *++out = ((*(float*)&ifval) - 3.0f);

        // update counter
        counter ++;
    }
```

Allan Herriman's analysis and illustrated treatise on the Voss-McCartney algorithm

Allan sent me two documents analysing the Voss approach to creating pink noise. Also noted here is an idea of mine for making the power distribution of the algorithm more closely approach the ideal.

The first was a three page **Excel spreadsheet** graphing the average 1024 FFTs of noise, each of them 524,288 points. That file as a 351k .zip file is available here: allan-1/graph.zip . One of its graphs is shown at the end of this page. Allan's notes on this spreadsheet file are: allan-1/graph-zip.readme.txt .

The second one was a Word file analysing theoretically how the multiple noise sources add up. I have converted this to HTML (with Word 97), so you can read **Allan's extensively illustrated treatise** here **allan-2/spectrum2.html** One graph from this is shown at the start of this page. Note that what Allan refers to as "row 0" for the white noise on each sample is what I refer to as "row -1" in my ASCII diagram above.

Be sure to read the above treatise!!!

Allan also contributed the following material on whether pink noise is "stationary" or not. One day I hope to be able to understand DSP to this level. For now, I am reproducing what he wrote verbatim.

> I think that to keep the academics happy we need a statement somewhere
> that true (wideband) pink noise is not stationary. A proof of this was
> given in comp.dsp a while back.
>
> http://www.deja.com/getdoc.xp?AN=438260245&fmt=text
>
> (Jeff Schenck's posting is archived here as: allan-1/stationary.html .)
>
> The problem stems from having a constant power per octave over an infinite
> number of octaves.
>
> One of the implications is that we can't make true pink noise by filtering
> white noise.
>
> OTOH, if we are only interested in accuracy down to some lower cutoff
> frequency, then the noise *is* stationary, and we *can* generate it by
> filtering white noise. This will generally be the case for audio
> applications.
>
> Note that the Voss algorithm is quite happy to sum an infinite number of
> octaves with only two random numbers per sample (you need infinite memory

> for this though), and thus isn't subject to the low frequency cutoff problem.

I asked him what "non stationary" meant, and he replied:

> From O+S:
> "... these probability distribution functions may be a function of the time index n. In the case where all the probability functions are independent of a shift of time origin, the random process is said to be stationary."
>
> In other words, the statistics of the signal don't change with time.
>
> It turns out that random processes with PSDs proportional to 1/f^a are stationary for a < 1. Pink noise represents the a = 1 case. Brown (or red?) noise represents the a = 2 case and is also not stationary.

Allan is quoting from one of the "Bibles of DSP" as mentioned at the comp.dsp FAQ site (see the references section below): A. V. Oppenheim and R. W. Schafer, Digital Signal Processing, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975. ISBN 0-13-214635-5.

Reducing the ripple in the algorithm's frequency response

There is some ripple in the frequency response of the Voss algorithm - as shown in the graphs at the top and bottom of this page. These reductions in the ripple of the algorithm's frequency response wouldn't be needed for my musical applications, but they might be useful for generating "analytical grade" pink noise for scientific purposes.

Allan suggests ways of reducing this.

> The ripple in Voss' generator can be reduced to half this level (to about 1/2 dB) by summing the outputs of two pink noise generators, one running at 1/sqrt(2) (about 70%) of the sample rate of the other. Odd sample rates like this might be a little tricky to implement, and would only be worthwhile for applications like test instruments (where you really care about flatness), etc.

His note on using similar techniques for reducing ripple in the filtered white noise approach is listed above in the filtering section.

I have another idea for reducing the ripple in the response: frequency modulate each row to lower octaves on a random basis! As I wrote to Allan:

> To implement random downwards frequency modulation for each row, for each time that the row is to be updated with a new random number, use a random function to decide whether to do so or to leave it unchanged.
>
> If a single update was missed, then that row's value would stay the same for twice as long as usual.
>
> In practical terms, if we want the probability of updating to be 50%, then, we keep the loop counter counting, but before calling the code to get a new random number and update the appropriate row, we look at a bit of a previously obtained random number and only call the code if the bit == 1.
>
> I would not like to write a formula for the frequency response of the results! I believe that by doing a drastic, noisy, frequency-modulation of each row, those distinctive curves in its frequency response will be scattered rather nicely and so the sum of all the row's frequency responses will be smoothed out.

> Overall, the frequency of the resulting noise would move downwards, by some factor which I couldn't be bothered thinking about right now. Therefore, we might want to add a lower level of white noise (my "row -1" to compensate for this.

On average, I guess this noise would be about 70% the frequency of the original.

- 50% of the time it would be the normal frequency - through doing the normal row update.
- 25% of the time it would be an octave down - through missing a single row update. (However, this covers two row sample times, so my "25%" should be higher?)
- 12.5% of the time it would be 1.5 octaves down - through missing two updates, and so running at 1/3 the normal sample rate.
- 6.25% of the time it would be 2 octaves down - through missing three updates.

The pattern continues, but these are for longer and longer times, so calculating the exact change to the power distribution is left as an exercise to the reader. (Allan had to get back to some other responsibilities for the time being.)

2006-03-27 Please see Larry Trammell's (RidgeRat's) "Stochastic Voss-McCartney algorithm" details below..

References to other material

Here are a few references to other material and web sites. Please let me know if any of these links no longer works. In some cases I have copies of the pages and software and will put them on my site if the original pages cannot be found anywhere.

There is a widely known article in Scientific American.

Stephan M. Sprenger (<sms@xxxx.prosoniq.com>, http://www.prosoniq.com/) wrote in Music-DSP on 30 June 1998:

> The original algorithm has been suggested by Voss in M. Gardner, "White and Brown Music, Fractal Curves and One-Over-f Fluctuations", Scientific American, 1978. It doesn't yield exact 1/f noise though, but for most practical applications it comes close enough.

Searching AltaVista for "voss and noise and gardner" I came across a site on chaos site http://usuarios.intercom.es/coclea/bennett.htm which mentioned another article by Voss:

> Voss, Richard F. and John Clarke. " '1/f noise' in Music: Music From 1/f Noise." J. Acoust. Soc. Am. 63(1) (1978): 258-261.

JASA is a fascinating journal. Everything from the neurochemistry of a gnat's auditory system to the propagation of sound waves in water across the Pacific Ocean. I will try to find these articles. Apparently the suggested algorithm is adding successive noise sources where each source is an octave below the previous one, and all have the same level. The

implementaion of this is discussed in Thomas Hudson's C++ code above.

Wentian Li's bibliography also lists another article: R.F. Voss and J. Clarke, "1/f noise in music and speech", Nature,258, 317-318 (1975).

On the Music-DSP list (25 August 1999) Chuck Cooper <plangent@xxxx.mediaone.net> wrote:

> About generating 1/F noise directly: There's an algorithm in "Computer Music -- Synthesis, Composition and Performance" by Charles Dodge and Thomas Jerse. Page 290 in the more recent edition. It's a pretty short loop. It picks one of N integers (N= a power of 2) with a 1/F distribution.

Wentian Li's bibliography on 1/f noise: http://linkage.rockefeller.edu/wli/1fnoise/ .

A good site for FAQ's DSP code and DSP development software is **DSP-Guru:** http://www.dspguru.com/ .

Denis Matignon has a site on noise which is related to pink-noise. http://tsi.enst.fr/~matignon/ I don't understand this field of fractional differential systems theory, but I think it relates to noise in servo systems which has a power distribution at a different slope from -3dB/octave. (Explanatory material at this site was not viewable with Netscape due to a missing style sheet, so use a less fussy browser instead.)

The FAQ for Usenet newsgroup **comp.dsp** is at: http://www.bdti.com/faq/ .

The web site for the **Music-DSP mailing list** is http://shoko.calarts.edu/~glmrboy/musicdsp/music-dsp.html .

(New April 2000) An article on pink noise, including an analogue filter circuit, by **Don Morgan** for *Embedded Systems* magazine. http://www.embedded.com/2000/0003/0003spectra.htm .

2006-03-27 **Larry Trammell's** (RidgeRat's) **"Stochastic Voss-McCartney algorithm": http://home.earthlink.net/~ltrammell/tech/pinkalg.htm** . He announced it on the MusicDSP mailing list: http://aulos.calarts.edu/pipermail/music-dsp/2006-March/thread.html . There is some interesting discussion on the list, which I have not attempted to summarise here - **please see the archives.**

2007-01-22 **Larry Trammell's** (RidgeRat's) new approach, which is faster and smoother: **http://home.earthlink.net/~ltrammell/tech/newpink.htm** .

2006-03-27 **Nicolas Brodu** has a paper using the Voss-McCartney algorithm with the highly respected Mersenne Twister PRNG:

http://arxiv.org/ftp/nlin/papers/0511/0511041.pdf
Real-time update of multi-fractal analysis on dynamic time series using incremental discrete wavelet transforms
Nicholas Brodu November 2005
GPL V2 C++ source code: http://nicolas.brodu.free.fr./en/programmation/incremfa/index.html .

2006-03-27 **Andrew Simper** has some C++ code for pink and brown noise, based on algorithms discussed here:

http://vellocet.com/dsp/noise/VRand.html

2006-03-27 It can be good to search for references to this page - you may find source code and other interesting stuff:

http://www.google.com/search?as_epq=www.firstpr.com.au%2Fdsp%2Fpink-noise
http://www.google.com/search?as_lq=www.firstpr.com.au%2Fdsp%2Fpink-noise%2F
http://scholar.google.com/scholar?as_epq=www.firstpr.com.au%2Fdsp%2Fpink-noise%2F

Please let me know if you find something I should add to this page.

2010-03-12 A good source of DSP information is **http://www.dsprelated.com** .

2011-03-20 Henning Thielemann's paper "Sampling-Rate-Aware Noise Generation" http://users.informatik.uni-halle.de/~thielema/Research/noise.pdf may be of interest. It concerns pink and white noise and methods of generating them which don't lead to unpleasant surprises if the sampling rate is changed.

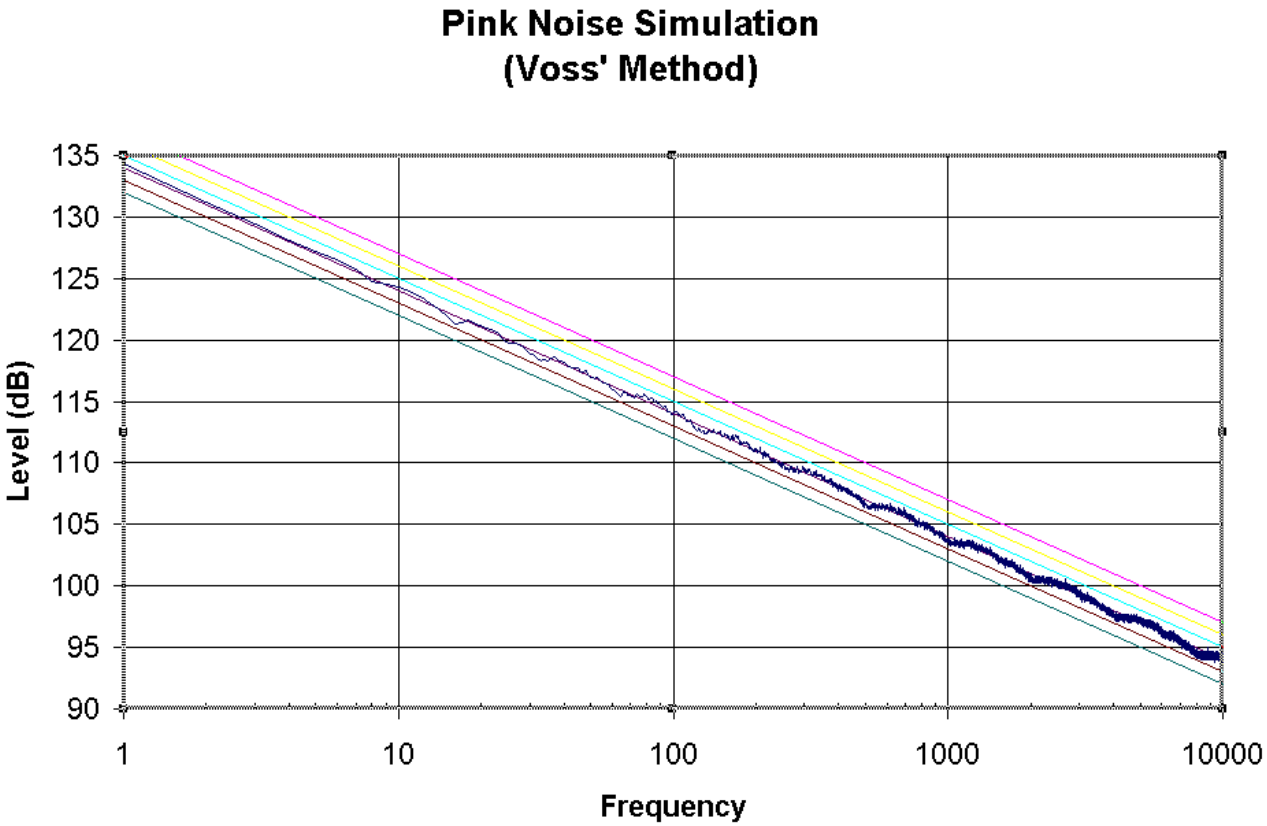
What next?

Please suggest improvements and additions for this page.

At some stage in the future, when I get my brain back into programming mode, I intend to write a pink noise ugen for Csound (and Quasimodo) for "k" and "a" rate. I can think of some ways to make a speedy algorithm by hard coding the first few rows of noise, writing it into a buffer (say 32 or 64 floats) which are subsequently read out until they are all goen and it is time to calculate a new set. This way, I think, we don't need to fuss over counting zeroes, except for the one sample of the 32 or 64 which changes according to the main counter. There was some discussion on the list of the merits of this, but it is hard to envisage the effects of caching and of how I would implement it. One view is that any use of buffers of pre-computed noise samples would be slower. However, maybe fast, hard-coded calculations filling a buffer, and that buffer being used in a loop to fill an array of "a" rate samples, looks attractive to me.

Thanks to James, Phil and Allan!

Here is a graph from Allan Herriman's spreadsheet. It is based on averaging 1024 FFTs of noise, each of them 524,288 points . . . . 10db per decade reduction in power is precisely what it should be.



Update history

See also minor updates at the beginning.

- 2007 January 22 Updated link to Larry's material.
- 2006 March 27 Link to Larry Trammell's Stochastic Voss-McCartney algorithm and to Nicholas Brodu's paper and source code.
- 2005 September 19 Added link to new Park-Miller-Carta PRNG page.
- 2003 July 10 Added link to pinkish opcode in Csound.
- 2000 April 11 Added link to Don Morgan's article.
- 1999 October 18 Removed Paul Kellet's two earlier filters, added Allan's graphical analysis of these two and Robert Bristow-Johnson's pinking filters into the main page. Also added Paul's description of how he devised the filters.
- 1999 October 17 Added updated pinking filter algorithms from Paul Kellet, but left his first two there, because Allan had analysed them.

- 1999 October 15 Added an updated allan-2/spectrum2.html and Allan's analysis of the pinking filters.
- 1999 September 20 Updated Paul Kellet's pinking filter.
- 1999 September 14 Everything revised in light of comments and contributions. Multiple revisions during the day. A completely new version of Allan's treatise added. Added my proposal for randomised frequency modulation of each row to reduce ripple.
- 1999 September 09 (9/9/99!) Page created. Allan's graph added.