The Color of Noise

Introduction

You've probably heard the sound of white noise. It's what we perceive as a hiss, and has a flat spectral content across all frequencies. But many other colors of noise exist, each with a different spectral "slope". Some emphasize lower frequencies, while others have a U-shape spectral weighting.

Common noise colors include pink, red, blue and violet. Each has different properties and applications, and are briefly described below.

White	• Equal power at each frequency
Pink	• Also known as 1/f noise or flicker noise • Spectral slope rate falls off at 3 dB per octave (or 10 dB per decade) • Equal power in proportional wide frequency bands, i.e. 10 Hz to 20 Hz has the same power as 10 kHz to 20 kHz • Similar to the human perception of sound, and mimics the western music scale • Better represents phenomena like wind, waves and heartbeats • Shown to improve sleep and memory
Red	• Also known as Brownian noise, because it mimics the noise of a Brownian walk • Spectral slope rate falls off at 6 dB per octave (20 dB per decade) • Emphasizes lower frequencies
Blue	• Spectral slope rate increases at 3 dB per octave (10 dB per decade) • Higher frequencies have greater spectral weighting • Cherenkov radiation is a natural example • Used in audio and visual dithering to reduce the perception of quantization
Violet	• Spectral slope rate increases at 6 dB per octave (20 dB per decade)

This application generates 1D pink, red, blue and violet noise, and produces periodograms for each.

Each noise sample is sonified so you can hear what it sounds like. Additionally, the sound can also be exported to a wave file for use in another tool (Maple will export high precision 32-bit and 64-bit wave files).

The approach described by Zhivomirov (2018) is used to generated colored noise. Briefly,

- white noise is generated and transformed to the frequency domain
- the spectral data is multiplied by a frequency-dependent factor
- the data is transformed back to the time domain

References:

- ZHIVOMIROV, H. 2018. A Method for Colored Noise Generation. Romanian Journal of Acoustics and Vibration. 15, 1 (Aug. 2018), 14-19.
- Hristo Zhivomirov (2020). Pink, Red, Blue and Violet Noise Generation with Matlab (https://www.mathworks.com/matlabcentral/fileexchange/42919-pink-red-blue-and-violet-noise-generation-with-matlab), MATLAB Central File Exchange. Retrieved April 8, 2020.

Parameters, Packages and Other Data

This loudspeaker component is needed to play the generated sound

```
> restart:
  with( SignalProcessing ):
  with( AudioTools ):
```

Signal length and sample rate

```
> signal_length := 2 ^ 15:
  Fs := 44100:
```

Plot style for periodograms

```
> dark_gridlines :=
   background   = black
  ,thickness    = 0
  ,axes         = frame
  ,axis[ 1 ]    = [ gridlines = [ 10, linestyle = dot, color = grey
  ], mode = log ]
  ,axis[ 2 ]    = [ gridlines = [ 10, linestyle = dot, color = grey
  ] ]
  ,font         = [ Arial ]
  ,size         = [ 600, 400 ]:
```

Procedure to Generate Colored Noise

This procedure is based on the algorithm in Zhivomirov (2018)

```
> ColoredNoise := proc( signal_length::integer, type::identical(
  "pink", "red", "blue", "violet" ) )
     #signal length must be an even integer
```

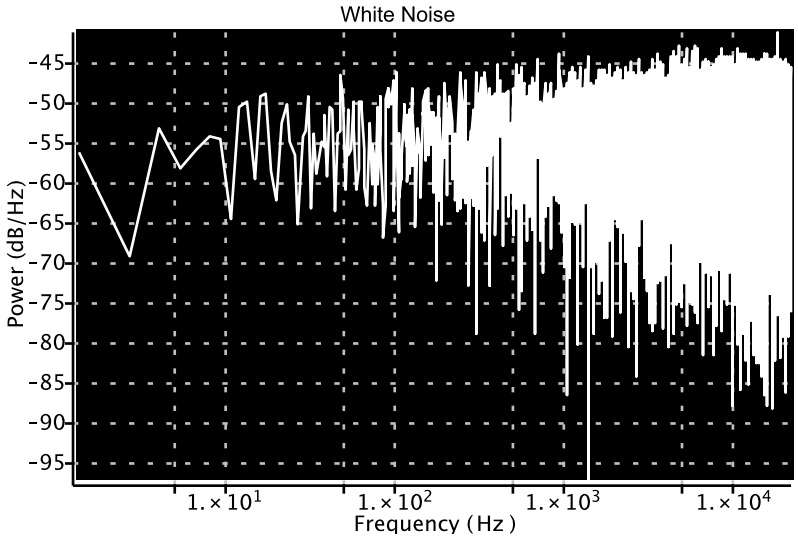
```
   local x, X, k, y:

   #Generate white noise
   x := LinearAlgebra:-RandomVector(signal_length, generator=-1.
.1., datatype  = float[8]):

   #Convert to frequency domain and only consider first half plus
1 (because the FFT is mirrored around the center)
   X := SignalProcessing:-FFT(x) *~ sqrt( signal_length ):
   X := X[ 1.. signal_length / 2 + 1 ]:

   # indicies for frequencies
   k := Vector( signal_length / 2 + 1, i -> i, datatype = float[ 8
] ):

   #Scale the frequency domain with respect to the frequency index
   if   type = "pink"   then X := X /~ sqrt~( k ):
   elif type = "red"    then X := X /~ k:
   elif type = "violet" then X := X *~ k:
   elif type = "blue"   then X := X *~ sqrt~( k ):
   end if:

   #Reconstruct the entire spectrum but remove the DC and Nyquist
components
   X := ArrayTools:-Concatenate( 2, X, conjugate~( ArrayTools:-
Reverse( X[ 2 .. -2 ] ) ) ):

   #Convert back to the time domain
   y := Re~( SignalProcessing:-InverseFFT( X /~ sqrt(
signal_length ) ) ):

   #Ensure mean of 0 and standard deviation of 1
   y := ( y -~ Mean( y ) ):
   y := y /~ StandardDeviation( y ):

   return y:

end proc:
```

Generate Colored Noise

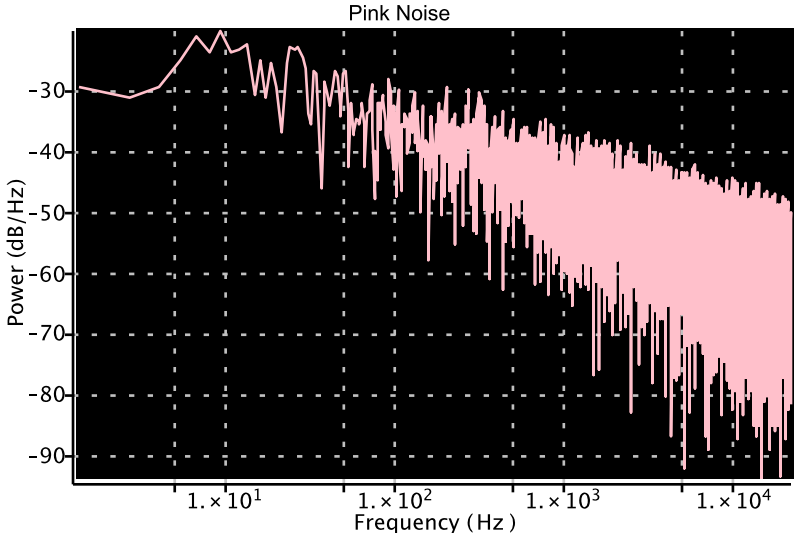
```
> whiteNoise  := LinearAlgebra:-RandomVector( signal_length,
  generator = -1 .. 1., datatype  = float[ 8 ] ):


> pinkNoise   := ColoredNoise( signal_length, "pink"   ):
  redNoise    := ColoredNoise( signal_length, "red"    ):
  blueNoise   := ColoredNoise( signal_length, "blue"   ):
  violetNoise := ColoredNoise( signal_length, "violet" ):
```

Spectral Analysis

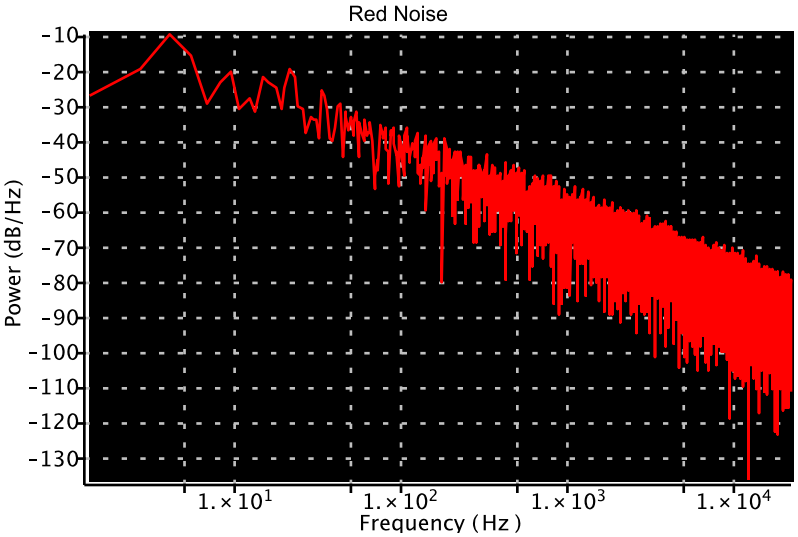
```
> Periodogram( HammingWindow( whiteNoise), powerscale = "dB/Hz",
  samplerate = Fs, title = "White Noise", color = white,
  dark_gridlines)
```




```
> Periodogram( HammingWindow( pinkNoise), powerscale = "dB/Hz",
  samplerate = Fs, title = "Pink Noise", color = pink,
  dark_gridlines )
```

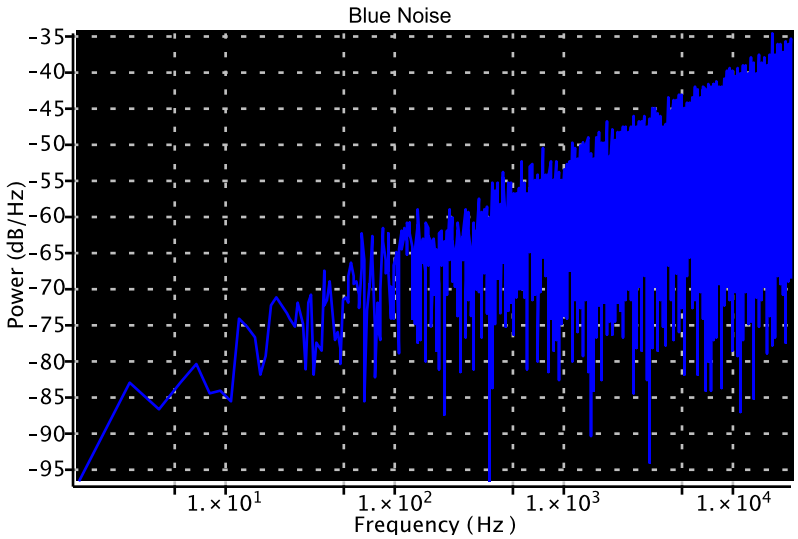



```
> Periodogram( HammingWindow( redNoise ),powerscale = "dB/Hz",
  samplerate = Fs, title = "Red Noise", color = red, dark_gridlines)
```

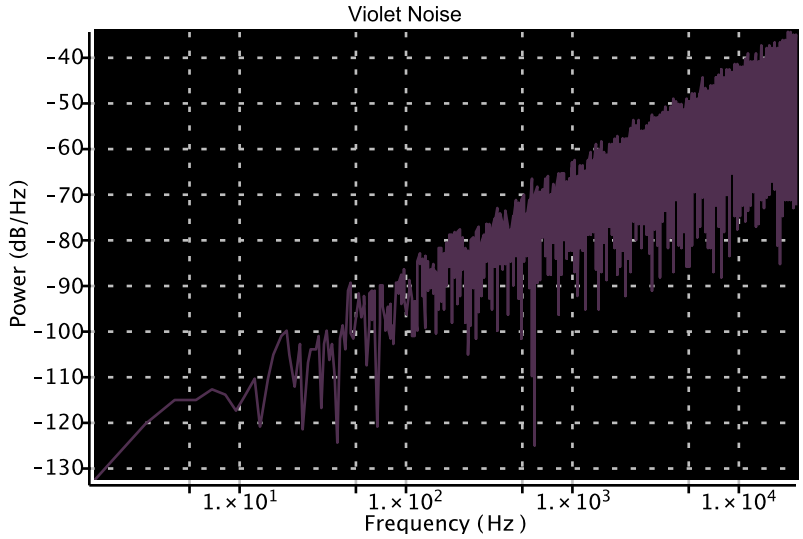



```
> Periodogram( HammingWindow( blueNoise ), powerscale = "dB/Hz",
```

```
samplerate = Fs, title = "Blue Noise", color = blue,
dark_gridlines)
```




```
> Periodogram( HammingWindow( violetNoise ), powerscale = "dB/Hz",
  samplerate = Fs, title = "Violet Noise", color = violet,
  dark_gridlines)
```

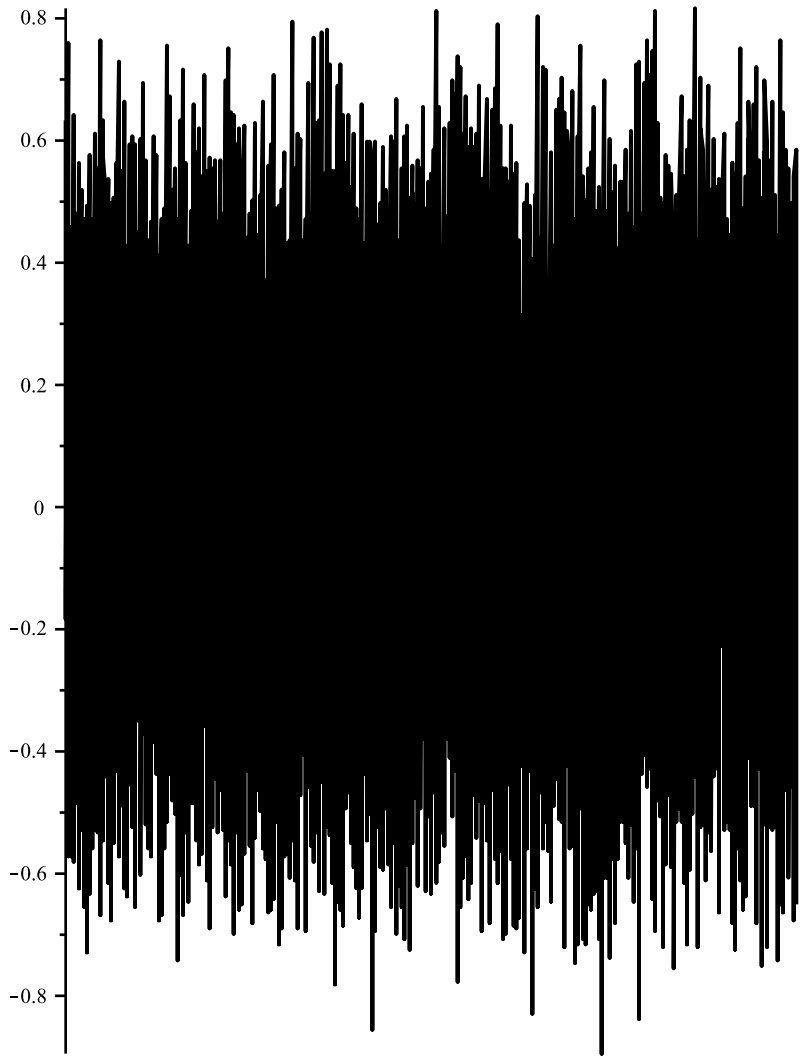


Sonification

Click the play button below each waveform to listen to the sound.

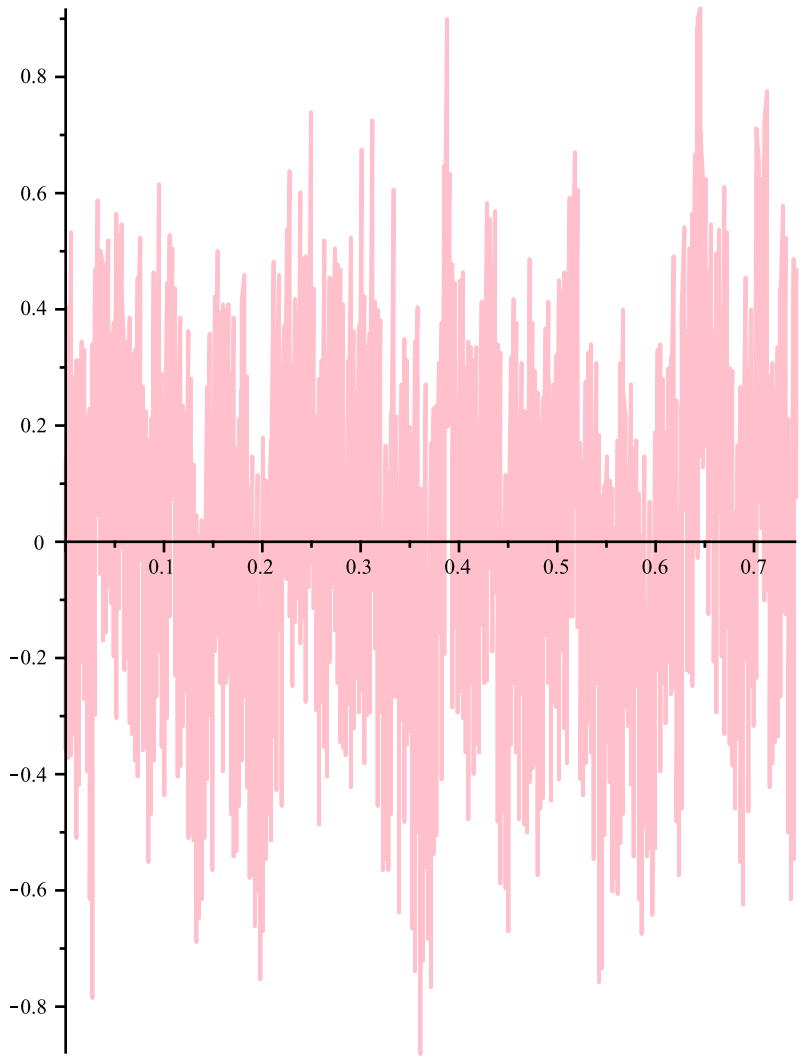
White noise

```
> Preview( Normalize( Create( whiteNoise, rate = Fs ), offset =
  scale), output = embed, color = black );
```



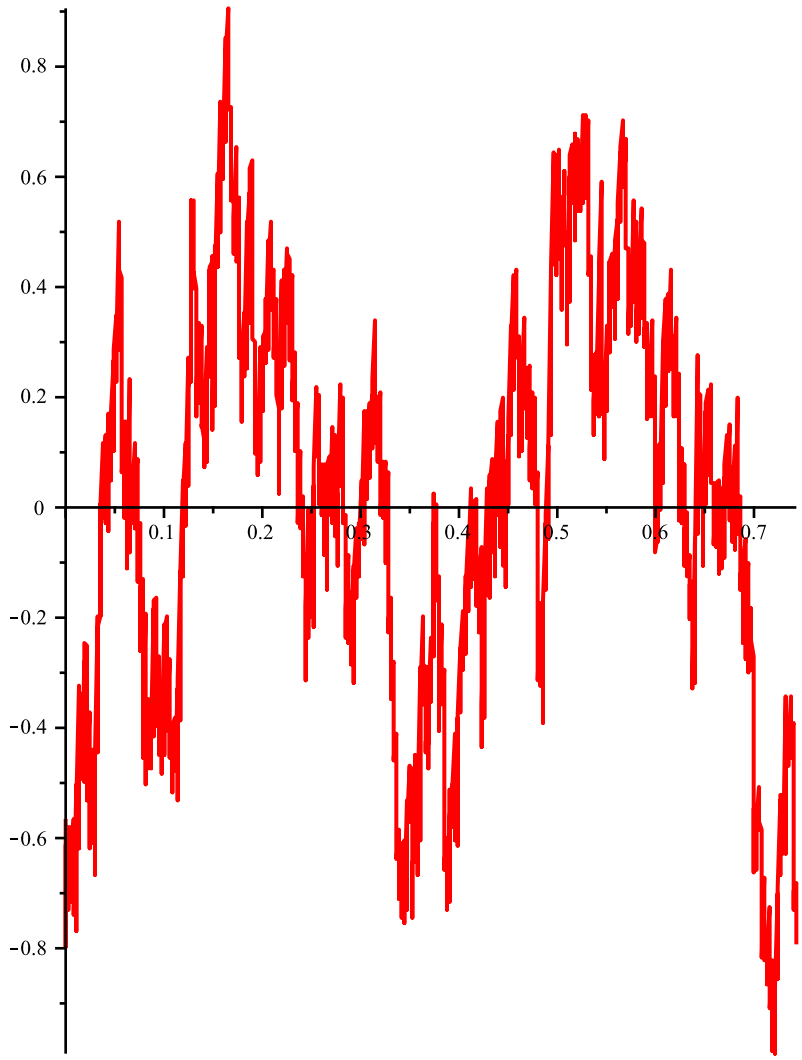
Pink noise

```
> Preview( Normalize( Create( pinkNoise, rate = Fs ), offset =
  scale), output = embed, color = pink );
```



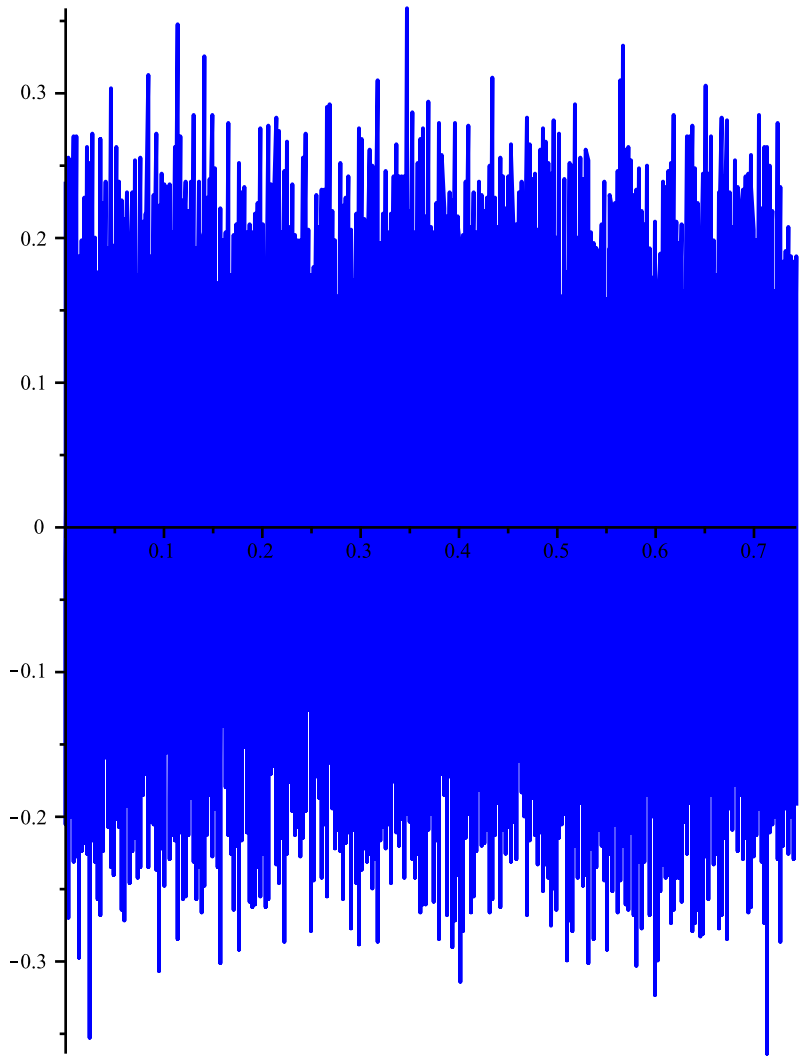
Red noise

```
> Preview( Normalize( Create( redNoise, rate = Fs ), offset =
  scale), output = embed, color = red )
```



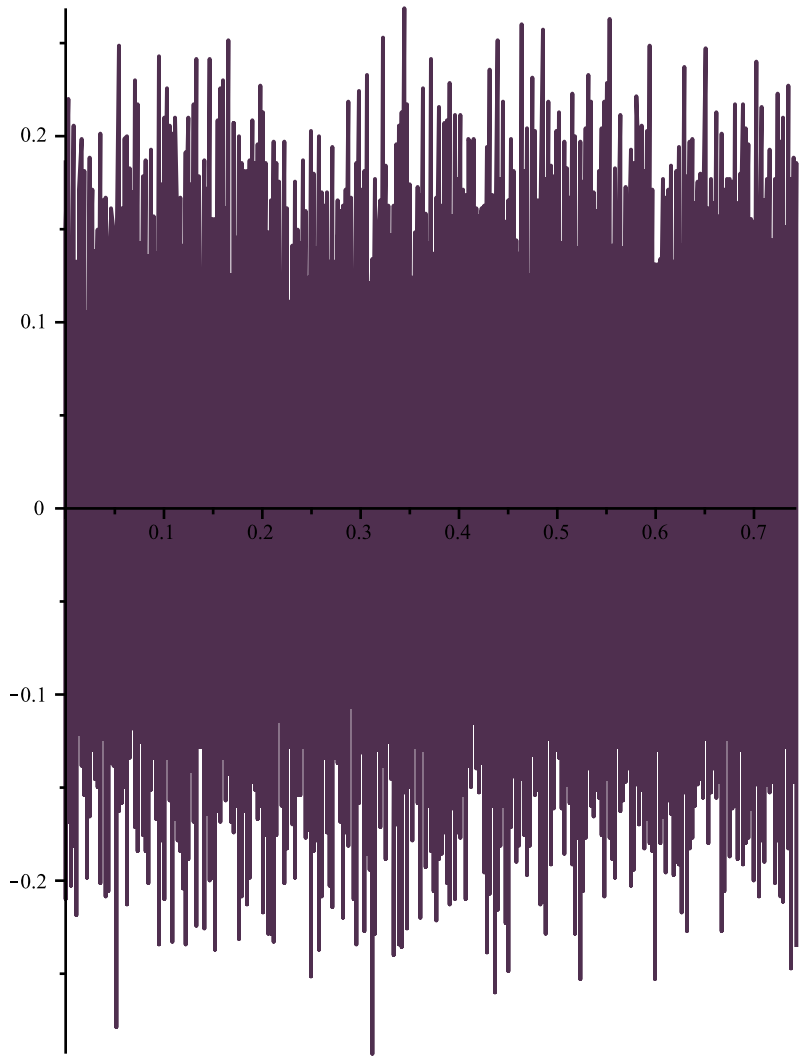
Blue noise

```
> Preview( Normalize( Create( blueNoise, rate = Fs ), offset = scale
  ), output = embed, color = blue )
```



Violet noise

```
> Preview( Normalize( Create( violetNoise, rate = Fs ), offset =
  scale ), output = embed, color = violet )
```


0.2
0.1
0
−0.1
−0.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7