Signal Analysis

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

Signal Processing with Free Software : Practical Experiments
F. Auger

Octave Spectrogram Function

Function File: **specgram** (x)
Function File: **specgram** (x, n)
Function File: **specgram** (x, n, Fs)
Function File: **specgram** (x, n, Fs, window)
Function File: **specgram** (x, n, Fs, window, overlap)
Function File: [S, f, t] = **specgram** (…)

https://octave.sourceforge.io/signal/function/specgram.html

Input and Output Arguments

x : the signal x.

n : the size of overlapping segments (default: 256)

fs : specifies the sampling rate of the input signal

window: specifies an alternate window (default: hanning)

overlap : specifies the number of samples overlap (default: (window)/2)

S : the complex output of the FFT, one row per slice

f : the frequency indices corresponding to the rows of S

t : the time indices corresponding to the columns of S.

https://octave.sourceforge.io/signal/function/specgram.html

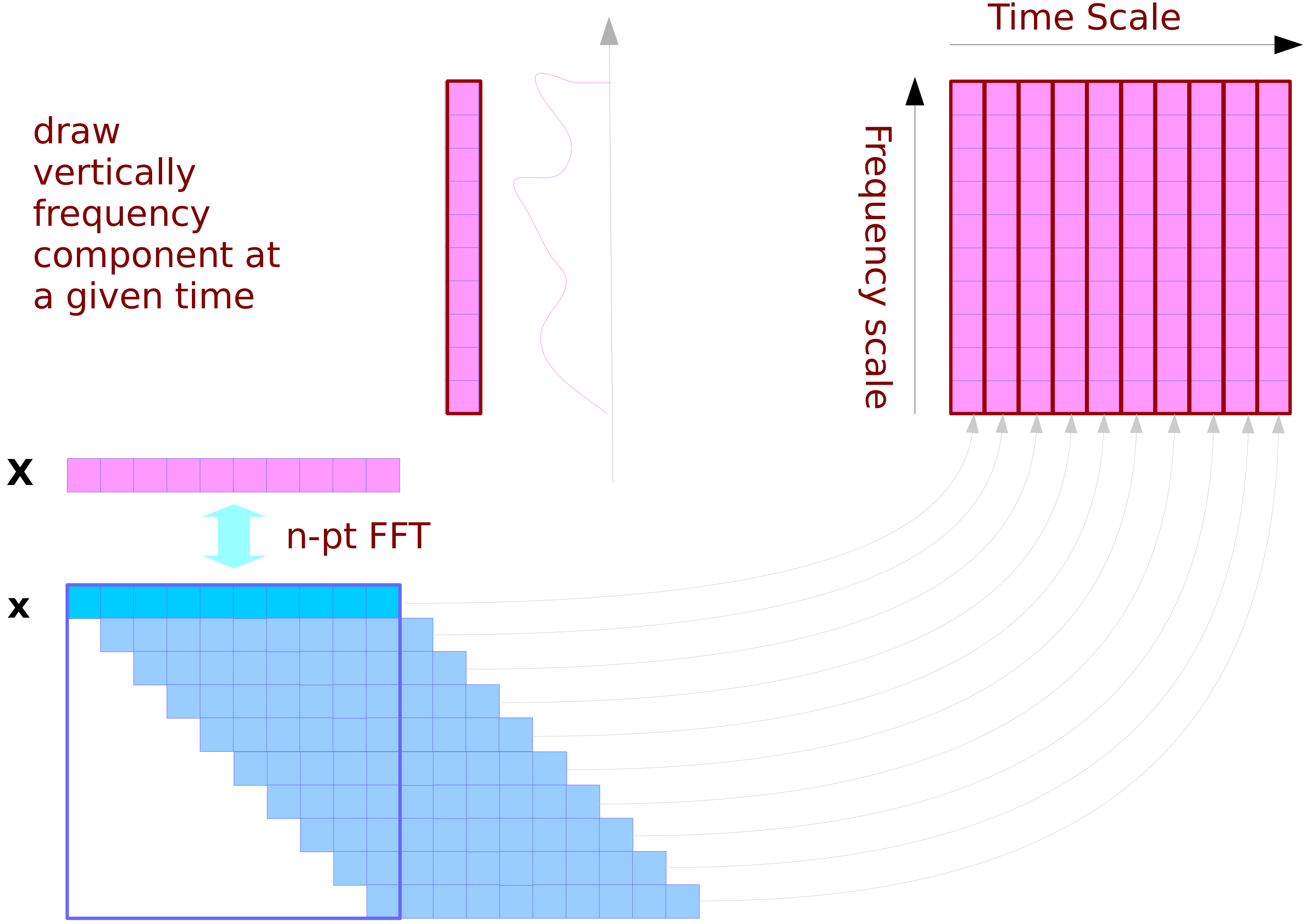
Spectrogram Opertions

- the signal is chopped into overlapping segments of length **n**
- each segment is **windowed** and transformed by using the **FFT**
- if **fs** is given, it specifies the sampling rate of the input signal
- an alternate window to apply rather than the default of hanning (n)
- the number of samples overlap between successive segments

- if no output arguments are given,
- the spectrogram is displayed.
- otherwise, [S, f, t] will be returned

https://octave.sourceforge.io/signal/function/specgram.html

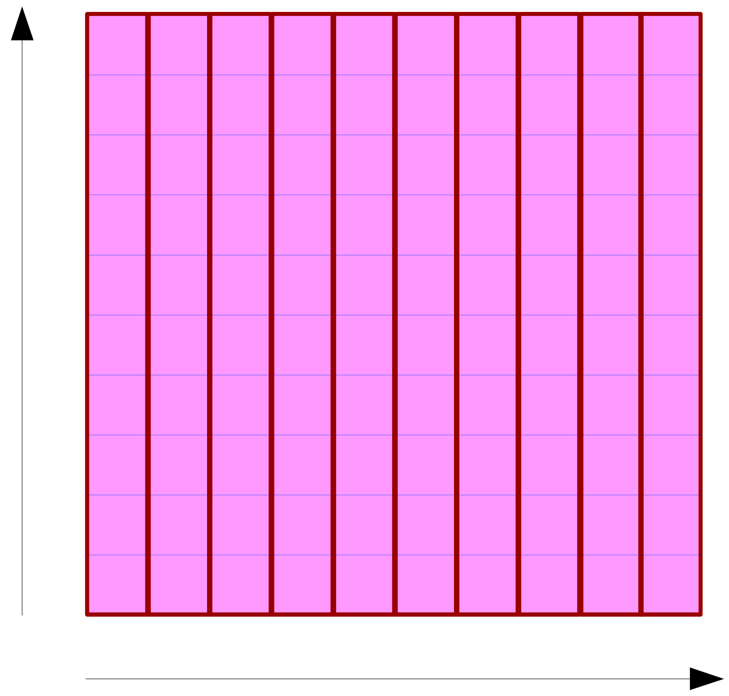
3D representation over Time Frequency Domain



Time and Frequency Resolutions

Frequency scale

Frequency Resolution
$= f_0 = f_s/n = 1/nT_s$



Time Scale

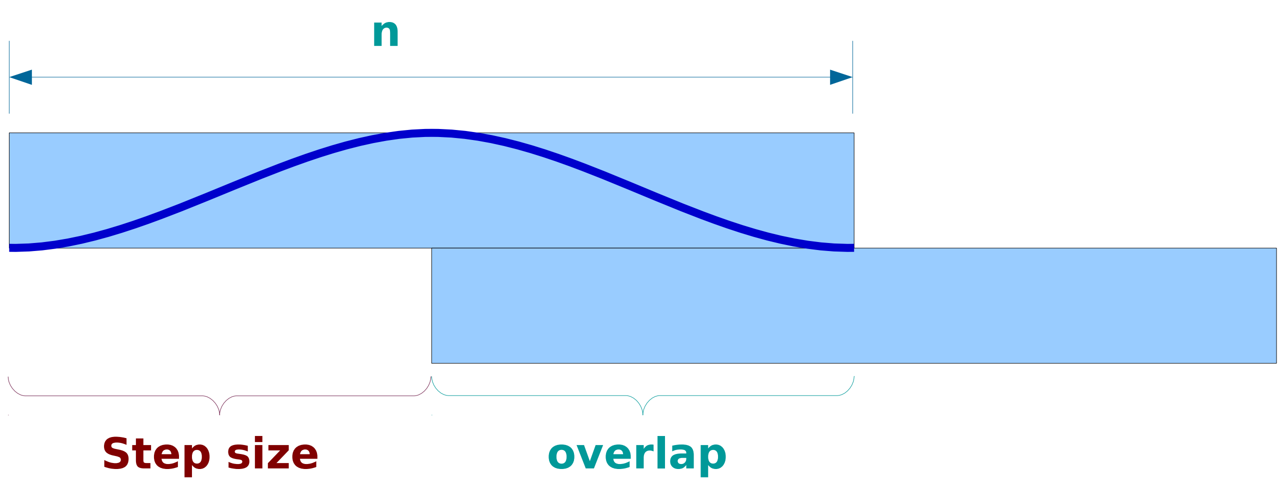
Time Resolution = step

Window Size

The choice of window defines the time-frequency resolution.

In speech for example,

- a wide window shows more harmonic detail
- a narrow window averages over the harmonic detail
  - shows more formant structure
- the shape of the window is not so critical
  - so long as it goes gradually to zero on the ends.



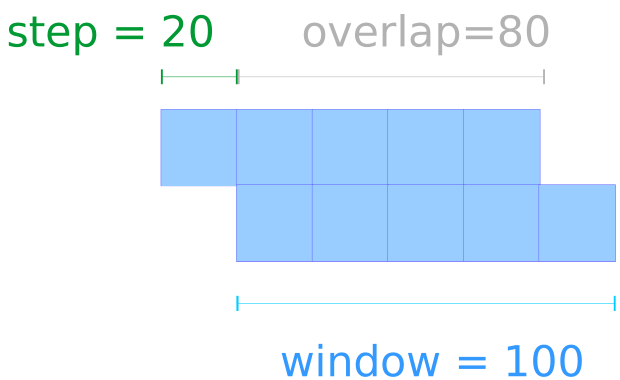
https://octave.sourceforge.io/signal/function/specgram.html

Step size

- window length minus overlap
- controls the horizontal scale of the spectrogram.

- gain a little bit, depending on the shape of your window
- as the peak of the window slides over peaks in the signal energy

- the range 1-5 msec is good for speech.

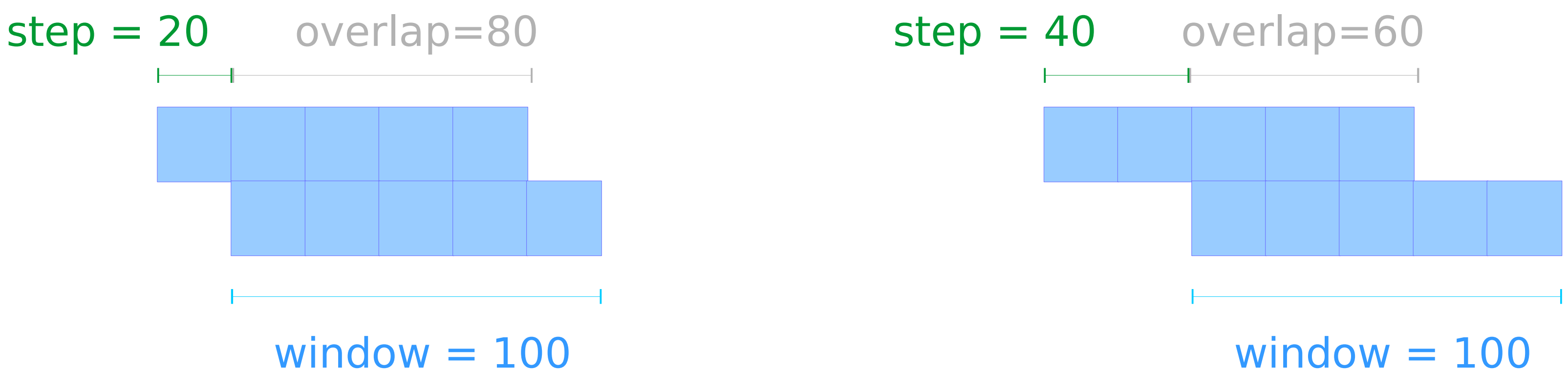


https://octave.sourceforge.io/signal/function/specgram.html

Step Size

Step size

- step size increase to compress.
- step size increase decrease to stretch
- increasing step size will reduce time resolution,
- decreasing it will not improve it much
    - beyond the limits imposed by the window size
- gain a little bit, depending on the shape of your window



https://octave.sourceforge.io/signal/function/specgram.html

FFT Length

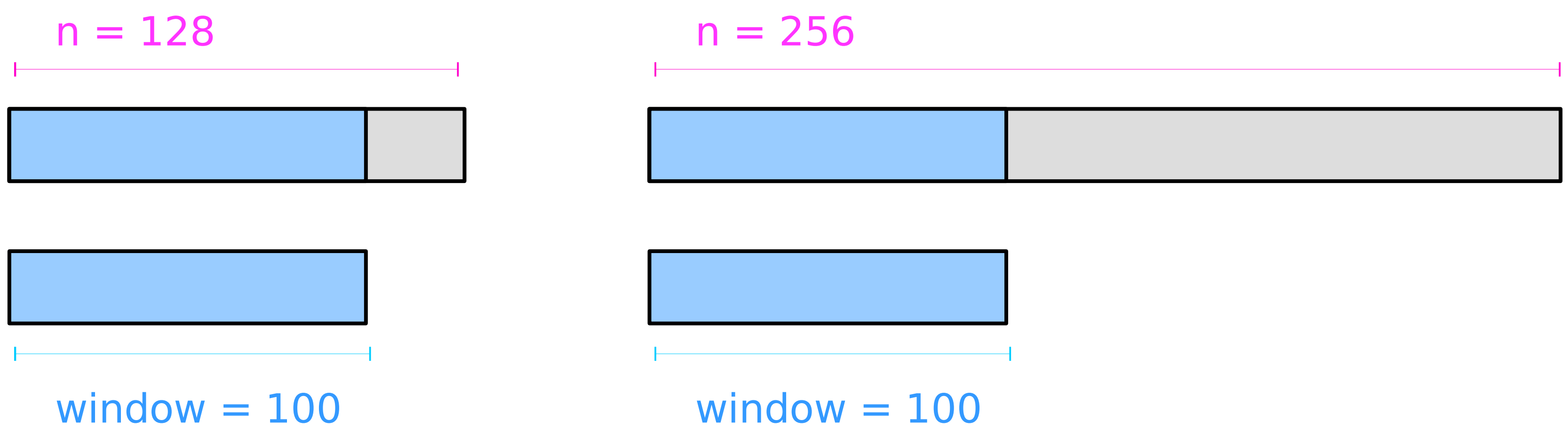
FFT length controls the <u>vertical</u> scale.

Selecting an <u>FFT length</u> *greater* than the <u>window length</u>

does not add any information to the spectrum

a good way to interpolate between frequency points

which can make for prettier spectrograms.



https://octave.sourceforge.io/signal/function/specgram.html

Dynamic Range

After you have generated the **spectral slices**
there are a number of decisions for displaying them.

- the phase information is discarded and
- the energy normalized:

```
S = abs(S); S = S/max(S(:));
```

then the dynamic range of the signal is chosen.

Since information in speech is well above the noise floor,
it makes sense to eliminate any dynamic range at the bottom end.
taking the max of the magnitude and
some minimum energy such as minE=-40dB.
Similarly, there is not much information in the very top of the range, so
clipping to a maximum energy such as maxE=-3dB makes sense:

```
S = max(S, 10^(minE/10));
S = min(S, 10^(maxE/10));
```

https://octave.sourceforge.io/signal/function/specgram.html

Frequency Range

The frequency range of the FFT is from 0
to the Nyquist frequency of one half the sampling rate. (Fs/2)

If the signal of interest is band limited,
you do not need to display the entire frequency range.

In speech for example, most of the signal is below 4 kHz,
so there is no reason to display
up to the Nyquist frequency of 10 kHz for a 20 kHz sampling rate.

In this case you will want to keep only the first 40% of the rows of the returned S and f.

More generally, to display the frequency range [minF, maxF],
you could use the following row index:

```
idx = (f >= minF & f <= maxF);
```

https://octave.sourceforge.io/signal/function/specgram.html

Color Map

then there is the choice of colormap.

A brightness varying colormap such as copper or bone
gives good shape to the ridges and valleys.

A hue varying colormap such as jet or hsv
gives an indication of the steepness of the slopes.

The final spectrogram is displayed in log energy scale
and by convention has low frequencies on the bottom of the image:

```
imagesc(t, f, flipud(log(S(idx,:))));
```

https://octave.sourceforge.io/signal/function/specgram.html

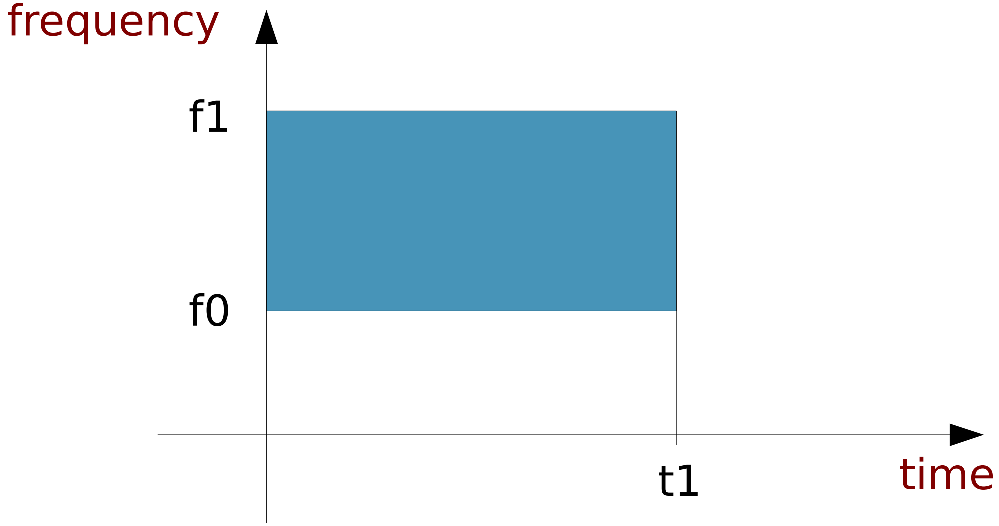
Example 1

```
x = chirp([0:0.001:2],0,2,500);  # freq. sweep from 0-500 over 2 sec.
Fs=1000;                          # sampled every 0.001 sec so rate is 1 kHz
step=ceil(20*Fs/1000);            # one spectral slice every 20 ms
window=ceil(100*Fs/1000);         # 100 ms data window
specgram(x, 2^nextpow2(window), Fs, window, window-step);

## Speech spectrogram
[x, Fs] = auload(file_in_loadpath("sample.wav")); # audio file
step = fix(5*Fs/1000);            # one spectral slice every 5 ms
window = fix(40*Fs/1000);         # 40 ms data window
fftn = 2^nextpow2(window);        # next highest power of 2
[S, f, t] = specgram(x, fftn, Fs, window, window-step);
S = abs(S(2:fftn*4000/Fs,:));     # magnitude in range 0<f<=4000 Hz.
S = S/max(S(:));                  # normalize magnitude so that max is 0 dB.
S = max(S, 10^(-40/10));          # clip below -40 dB.
S = min(S, 10^(-3/10));           # clip above -3 dB.
imagesc (t, f, log(S));                # display in log scale
set (gca, "ydir", "normal");      # put the 'y' direction in the correct direction
```

https://octave.sourceforge.io/signal/function/specgram.html

Chirp (1)

Function File: **chirp** (t)
Function File: **chirp** (t, f0)
Function File: **chirp** (t, f0, t1)
Function File: **chirp** (t, f0, t1, f1)
Function File: **chirp** (t, f0, t1, f1, form)
Function File: **chirp** (t, f0, t1, f1, form, phase)



https://octave.sourceforge.io/signal/function/chirp.html

Chirp (2)

Function File: **chirp** (t)
Function File: **chirp** (t, f0)
Function File: **chirp** (t, f0, t1)
Function File: **chirp** (t, f0, t1, f1)
Function File: **chirp** (t, f0, t1, f1, form)
Function File: **chirp** (t, f0, t1, f1, form, phase)

$$f(t) = \frac{(f_1 - f_0)}{(t_1 - 0)} \cdot t + f_0$$

Evaluate a chirp signal at time t.
A chirp signal is a frequency swept cosine wave.

t	vector of times to evaluate the chirp signal
f0	frequency at time t=0 [0 Hz]
t1	time t1 [1 sec]
f1	frequency at time t=t1 [100 Hz]
form	shape of frequency sweep
	'<u>linear</u>' f(t) = (f1-f0)*(t/t1) + f0
	'<u>quadratic</u>' f(t) = (f1-f0)*(t/t1)^2 + f0
	'<u>logarithmic</u>' f(t) = (f1-f0)^(t/t1) + f0
phase	phase shift at t=0

$$f(t) = (f_1 - f_0) \cdot \left(\frac{t}{t_1}\right) + f_0$$

$$f(t) = (f_1 - f_0) \cdot \left(\frac{t}{t_1}\right)^2 + f_0$$

$$f(t) = (f_1 - f_0)^{\left(\frac{t}{t_1}\right)} + f_0$$

https://octave.sourceforge.io/signal/function/specgram.html

Chirp (3)

Function File: **chirp** (**t**)
Function File: **chirp** (**t**, f0)
Function File: **chirp** (**t**, f0, t1)
Function File: **chirp** (**t**, f0, t1, f1)
Function File: **chirp** (**t**, f0, t1, f1, form)
Function File: **chirp** (**t**, f0, t1, f1, form, phase)

t	a time vector
f0	frequency at time t=0
t1	time t1
f1	frequency at time t=t1
form	shape of frequency sweep
phase	phase shift at t=0

Example

```
specgram(chirp([0:0.001:5]));                    # linear, 0-100Hz in 1 sec
specgram(chirp([-2:0.001:15], 400, 10, 100, 'quadratic'));

soundsc(chirp([0:1/8000:5], 200, 2, 500, "logarithmic"),8000);
```

If you want a different sweep shape f(t), use the following:
y = cos(2*pi*integral(f(t)) + 2*pi*f0*t + phase);

```
x = chirp([0:0.001:2],0,2,500);    # freq. sweep from 0-500 over 2 sec.
```

https://octave.sourceforge.io/signal/function/specgram.html

Example 1

```
x = chirp([0:0.001:2],0,2,500);  # freq. sweep from 0-500 over 2 sec.
Fs=1000;                         # sampled every 0.001 sec so rate is 1 kHz
step=ceil(20*Fs/1000);           # one spectral slice every 20 ms
window=ceil(100*Fs/1000);        # 100 ms data window
specgram(x, 2^nextpow2(window), Fs, window, window-step);
```

Fs = 1000 Hz = 1 kHz
Ts = 1/1000 sec = 1 msec

step = 20 msec
window = 100 msec

x = x
n = 2^nextpow2(100) = 2^128
Fs = 1000
window = 100
overlap = 100-20 = 80

https://octave.sourceforge.io/signal/function/specgram.html

Example 1

```
x = chirp([0:0.001:2],0,2,500);  # freq. sweep from 0-500 over 2 sec.
Fs=1000;                         # sampled every 0.001 sec so rate is 1 kHz
step=ceil(20*Fs/1000);           # one spectral slice every 20 ms
window=ceil(100*Fs/1000);        # 100 ms data window
specgram(x, 2^nextpow2(window), Fs, window, window-step);
```

Fs = 1000 Hz = 1 kHz
Ts = 1/1000 sec = 1 msec

step = 20 msec
window = 100 msec

2 sec   2 sec * 1000 samples /sec = 2000 samples

20 msec * 1 samples /msec = 20 samples

20 msec * Fs samples/sec / (1000 msec/sec)

https://octave.sourceforge.io/signal/function/specgram.html

Example 1

Fs = 1000 Hz = 1 kHz
Ts = 1/1000 sec = 1 msec

step = 20 msec : 20 samples
window = 100 msec : 100 samples

2000 samples = 96 steps * 20 samples /step + 80 samples
= (1920 + 80) samples

2 sec

2000 samples = 96 steps + 80 samples

20 msec * 1 samples /msec = 20 samples

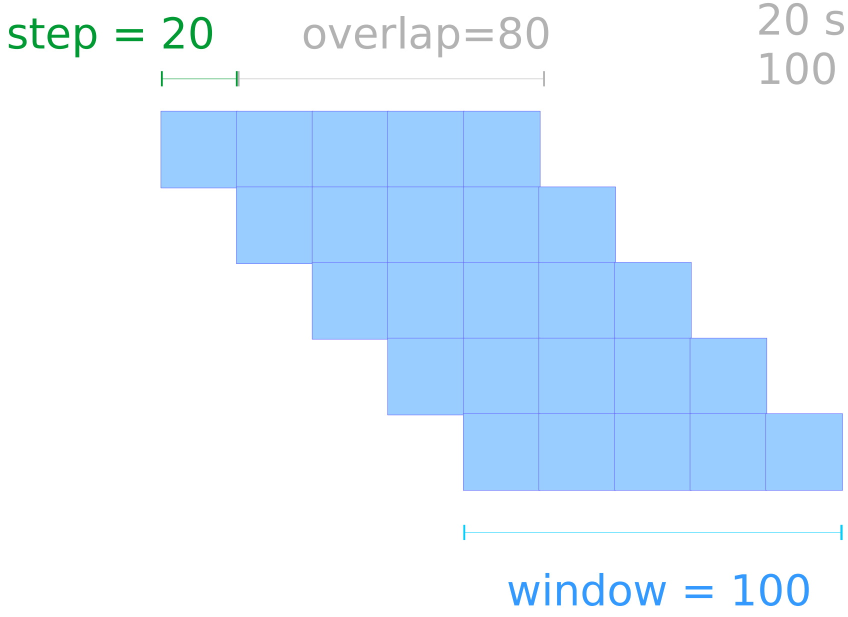
20 msec * Fs samples/sec / (1000 msec/sec)

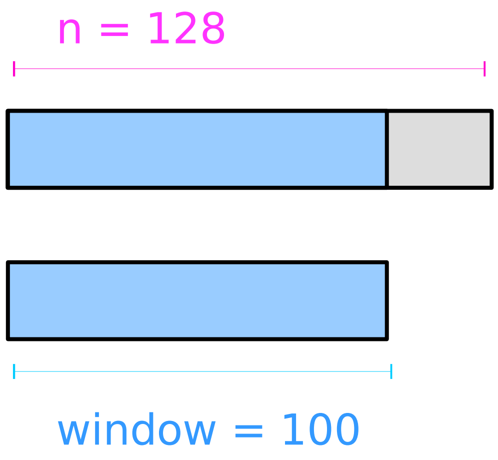
https://octave.sourceforge.io/signal/function/specgram.html

Example 1

```
x = chirp([0:0.001:2],0,2,500);   # freq. sweep from 0-500 over 2 sec.
Fs=1000;                          # sampled every 0.001 sec so rate is 1 kHz
step=ceil(20*Fs/1000);            # one spectral slice every 20 ms
window=ceil(100*Fs/1000);         # 100 ms data window
specgram(x, 128, Fs, 100, 80);
```

a sample : 0.001 sec = 1 msec
20 samples : 20 msec
100 samples : 100 msec





https://octave.sourceforge.io/signal/function/specgram.html

Example 2

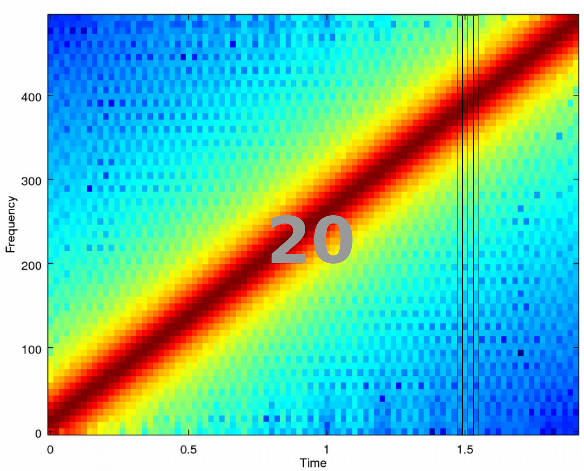
```
Fs=1000;
x = chirp([0:1/Fs:2],0,2,500);          # freq. sweep from 0-500 over 2 sec.
step=ceil(20*Fs/1000);                  # one spectral slice every 20 ms
window=ceil(100*Fs/1000);               # 100 ms data window

## test of automatic plot
[S, f, t] = specgram(x);
specgram(x, 2^nextpow2(window), Fs, window, window-step);
```

https://octave.sourceforge.io/signal/function/specgram.html

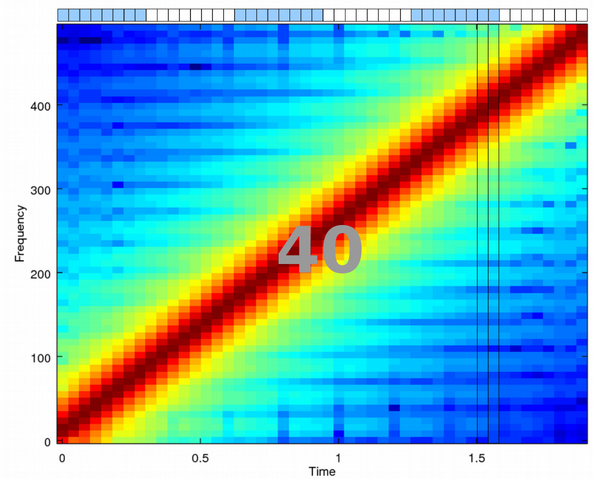
Example 2

https://octave.sourceforge.io/signal/function/specgram.html

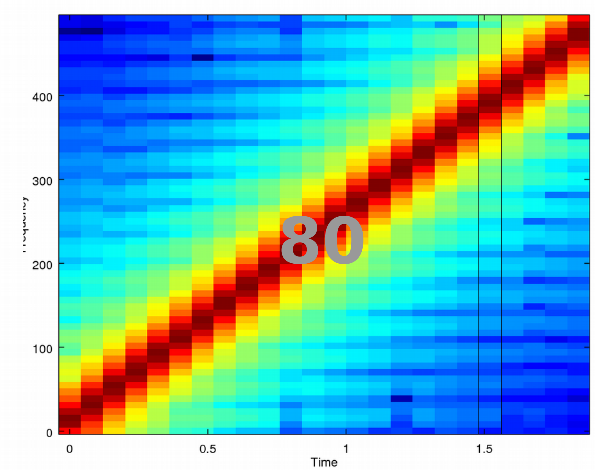
```
Fs=1000;
x = chirp([0:1/Fs:2],0,2,500);            # freq. sweep from 0-500 over 2 sec.
step=ceil(20*Fs/1000);                    # one spectral slice every 20 ms
window=ceil(100*Fs/1000);                 # 100 ms data window

## test of automatic plot
[S, f, t] = specgram(x);
specgram(x, 2^nextpow2(window), Fs, window, window-step);
```



step=20msec
96 steps



step=40msec
48 step



step=80msec
24 steps

References

[1] F. Auger, Signal Processing with Free Software : Practical Experiments