



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

LISTA FOURIER 2 - CAP-384

Leonardo Sattler Cassara

Lista de Exercícios apresentada aos professores Margarete Domingues e Luciano Magrini como parte da avaliação do curso CAP-384.

Repositório desta lista:
github/CAP-384

INPE
São José dos Campos
16 de outubro de 2020

PREFÁCIO

Os códigos desta lista utilizam a linguagem Python. As análises foram realizadas com a biblioteca `numpy`, em particular com a rotina `numpy.fft`, que implementa a transformada discreta de Fourier através de um algoritmo de transformada rápida de Fourier. As visualizações foram geradas com a biblioteca `matplotlib`. Todos os códigos e as imagens que eles geram estão organizados na pasta `scripts` do repositório deste manuscrito. Os arquivos estão separados por exercício conforme descrito abaixo.

- pasta **exercicio1**: contém scripts que geram sinais chirps e suas transformadas.
- pasta **exercicio2**: contém um script que implementa a transformada janelada de Fourier e gera os espectrogramas dos chirps através de seis funções janela.

Os trechos mais relevantes dos scripts deste repositório estão listados neste manuscrito. A correta compilação deste manuscrito depende das imagens e dos scripts presentes na pasta `scripts`. Os scripts Python podem ser executados de qualquer local. Os relatórios pertinentes aos Exercícios 3 e 4 estão em preparação e serão adicionados a este manuscrito em breve através de atualizações do seu repositório.

SUMÁRIO

	<u>Pág.</u>
Exercício 1	1
1.1	1
1.2	2
1.1	2
1.2	2
Exercício 2	7
2.a	7
2.b	16
2.c	22
Exercício 3	32
3.1	32
3.2	32
Exercício 4	33
4.1	33
4.2	33
4.3	33
4.4	33
4.5	33

Exercício 1

Neste exercício são graficados diferentes chirps e seus espectros de Fourier. Chirps são funções contínuas que podem representar sinais com as características de frequência crescente (*chirp-up*) ou descrescente (*chirp-down*) no tempo. Quatro chirps diferentes são aqui estudados: chirp gaussiano, linear, quadrático (ou square) e hiperbólico. Eles são explorados no contexto da Análise de Fourier.

A função de densidade espectral para esse exercício foi assim definida:

```
1 def psd(signal, sr=1, N=2048):
2     """
3         numpy.fft.fft:
4             When the input a is a time-domain signal and A = fft(a):
5                 . np.abs(A) is its amplitude spectrum;
6                 . np.abs(A)**2 is its power spectrum;
7                 . np.angle(A) is the phase spectrum.
8
9         f = signal
10        ft = fft.fft(f, N)
11        ft_shifted = fft.fftshift(ft)
12        aft = np.abs(ft)**2
13        aft_shifted = abs(ft_shifted)**2 # spectrum shifted
14        # shifting frequencies
15        freq = np.fft.fftfreq(N, d=sr) # sr = sampling rate
16        freqs = np.concatenate([freq[int(len(freq)/2):],[0]])
17        freqs = np.concatenate([freqs, freq[1:int(len(freq)/2)]])
18        #
19        return freqs, aft_shifted
```

1.1

Definição da função **chirp gaussiano**:

```
1 def gaussian_chirp(t, a=1., b=2*pi, c=6*pi):
2     return a * np.exp( - (b - 1.j * c) * (t**2) )
```

Gerando sinal chirp e seu espectro (variável global **N** também é usada na definição dos demais sinais):

```
1 N = 2048 # Number of bins (samples)
2 #-----
3 ti_1 = 0#-2 # initial time
4 tf_1 = 6#2 # final time
5 t_1 = np.linspace(ti_1, tf_1, N) # time bins
```

```

6 res_1 = abs(ti_1 - tf_1)/len(t_1) # resolution (sampling rate)
7
8 g_1 = gaussian_chirp(t_1) # chirp
9 x_1, ft_1 = psd(g_1, res_1, N) # frequency bins + psd of signal

```

1.2

Definição da função **chirp linear**:

```

1 def linear_chirp(t, a=1., b=1/4.):
2     return a * np.exp( 1.j * b * (t**2) )

```

Gerando sinal chirp e seu espectro:

```

1 ti_2 = 0#-6
2 tf_2 = 6
3 t_2 = np.linspace(ti_2, tf_2, N)
4 res_2 = abs(ti_2 - tf_2)/len(t_2)
5
6 g_2 = linear_chirp(t_2)
7 x_2, ft_2 = psd(g_2, res_2, N)

```

1.3

Definição da função **chirp quadrático ou square**:

```

1 def square_chirp(t, a=1., b=1/4.):
2     return a * np.exp( 1.j * b * (t**3) )

```

Gerando sinal chirp e seu espectro:

```

1 ti_3 = 0#-4
2 tf_3 = 6#4
3 t_3 = np.linspace(ti_3, tf_3, N)
4 res_3 = abs(ti_3 - tf_3)/len(t_3)
5
6 g_3 = square_chirp(t_3)
7 x_3, ft_3 = psd(g_3, .01)

```

1.4

Definição da função **chirp hiperbólico**:

```

1 def hyper_chirp(t, a=1., b=6*pi, c=2.3):
2     return a * np.cos( b / (c - t) )

```

Gerando sinal chirp e seu espectro:

```

1 ti_4 = 0
2 tf_4 = 6#2
3 t_4 = np.linspace(ti_4, tf_4, N)
4 res_4 = abs(ti_4 - tf_4)/len(t_4)
5
6 g_4 = hyper_chirp(t_4)
7 x_4, ft_4 = psd(g_4, res_4, N)

```

A Figura com os chirps e seus espectros foi assim gerada (Figura 1.1):

```

1 fig, ax = plt.subplots(4, 2, figsize=(9,7)) # (4 x 2) subplots
2 #----- Row 1
3 # Column 1
4 ax[0,0].set_title('Chirp Signals', size=15, loc='left')
5 ax[0,0].plot(t_1, g_1.real, 'r-', label='real')
6 ax[0,0].plot(t_1, g_1.imag, 'b--', label='imag')
7 ax[0,0].legend(bbox_to_anchor=(.6,1.55), loc="upper left")
8 ax[0,0].set_ylabel('Gaussian \nchirp', rotation=0, labelpad=25.,
9 size=12)
9 # Column 2
10 ax[0,1].set_title('Power Spectrum', size=15)
11 ax[0,1].plot(x_1, ft_1, 'k-')
12 #----- Row 2
13 # Column 1
14 ax[1,0].plot(t_2, g_2.real, 'r-')
15 ax[1,0].plot(t_2, g_2.imag, 'b--')
16 ax[1,0].set_ylabel('Linear \nchirp', rotation=0, labelpad=25., size
17 =12)
17 # Column 2
18 ax[1,1].plot(x_2, ft_2, 'k-')
19 #----- Row 3
20 # Column 1
21 ax[2,0].plot(t_3, g_3.real, 'r-')
22 ax[2,0].plot(t_3, g_3.imag, 'b--')
23 ax[2,0].set_ylabel('Square \nchirp', rotation=0, labelpad=25., size
24 =12)
24 # Column 2
25 ax[2,1].plot(x_3, ft_3, 'k-')
26 #----- Row 4
27 # Column 1
28 ax[3,0].plot(t_4, g_4.real, 'r-')
29 ax[3,0].plot(t_4, g_4.imag, 'b--')
30 ax[3,0].set_ylabel('Hyperbolic \nchirp', rotation=0, labelpad=25.,
31 size=12)
31 ax[3,0].set_xlabel('t', size=12)

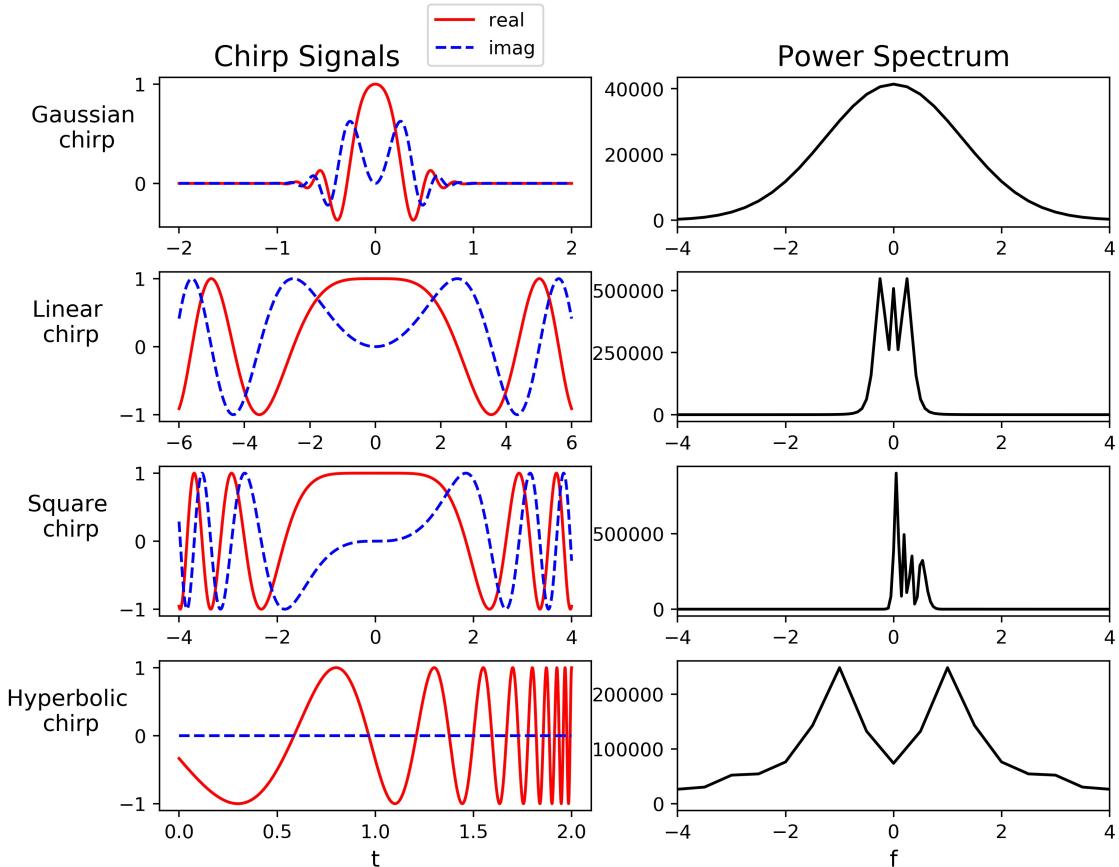
```

```

32 # Column 2
33 ax[3,1].plot(x_4, ft_4, 'k-')
34 ax[3,1].set_xlabel('f', size=12)
35 #----- End of individual-plot settings
36 for i in range(4):
37     # Setting same x_lim for all axes
38     ax[i,1].set_xlim(-4,4)
39 # Setting spacing between subplots
40 fig.subplots_adjust(left=None, bottom=None, \
41                     right=None, top=None, \
42                     wspace=.2, hspace=.3)
43 #----- Saving and showing plot
44 fig.savefig('TEST_chirps_psd.jpg', dpi=400, bbox_inches='tight')
45 plt.show()

```

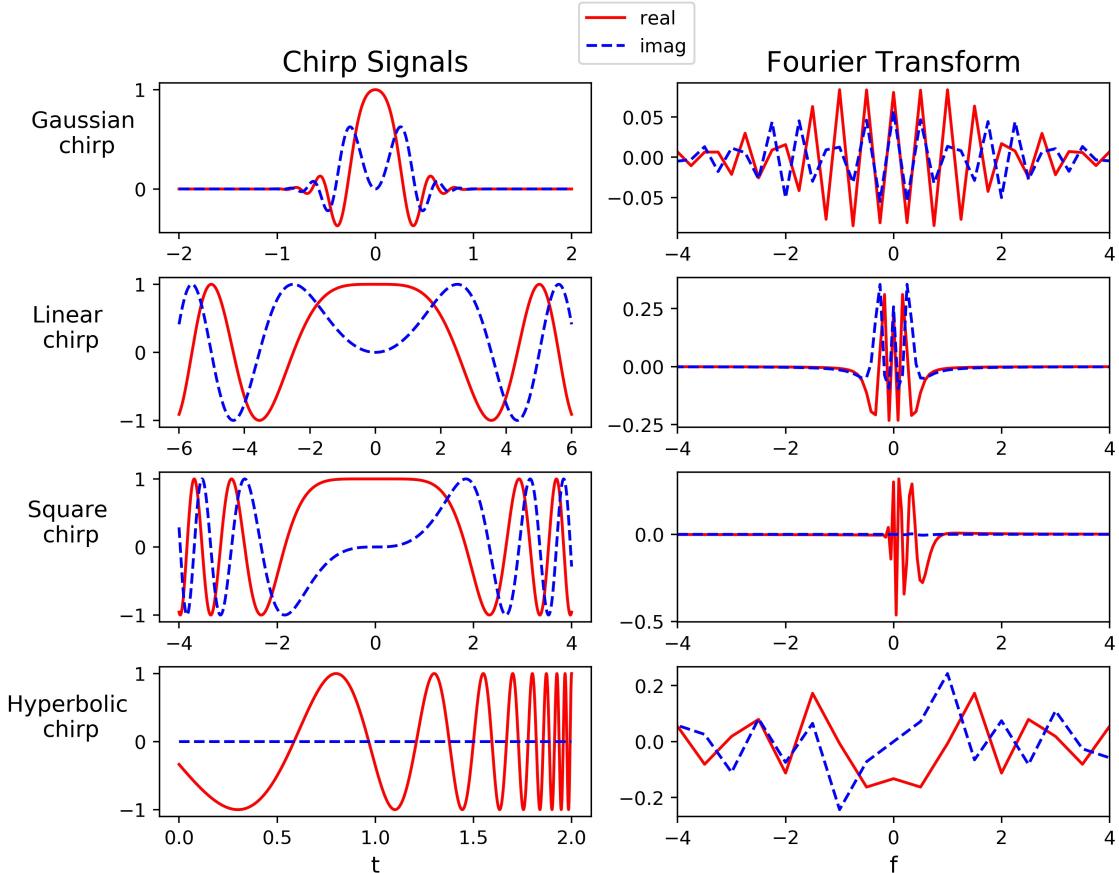
Figura 1.1: Sinais chirp do Exercício 1 e seus respectivos espectros de potência calculados via FFT. As partes reais (linhas cheias em vermelho) e imaginárias (linhas tracejadas em azul) dos chirps são exibidas.



O código do início desta seção pode ser modificado (função `psd`) de modo que seu

output seja a Transformada de Fourier. O resultado é exibido na Figura 1.2.

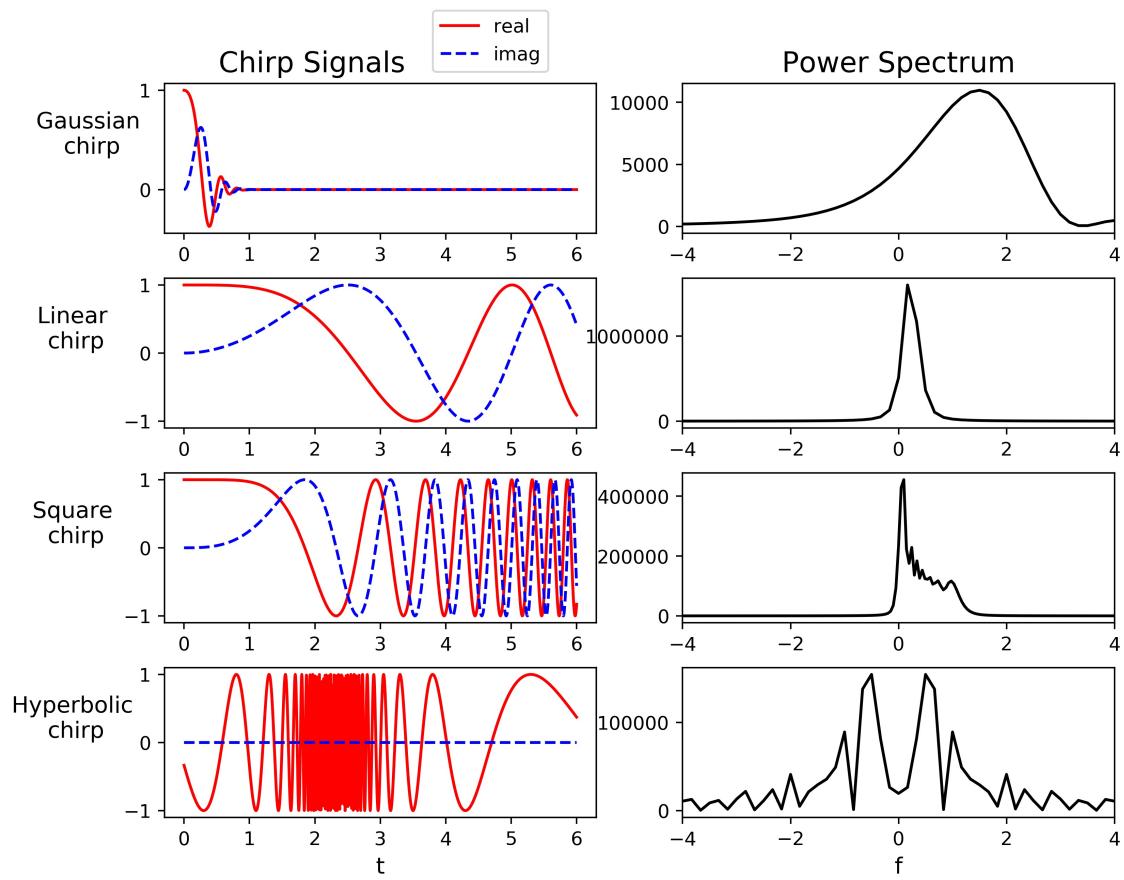
Figura 1.2: Sinais chirp e suas Transformadas de Fourier calculadas via FFT. As partes reais (linhas cheias em vermelho) e imaginárias (linhas tracejadas em azul) são exibidas.



A Figura a seguir exibe o comportamento dos chirps e suas transformadas para $t \in [0, 6]$:

Conclui-se das Figuras 1.1, 1.2 e 1.3 que os chirps possuem uma natureza distinta quanto à variação de suas frequências ao longo do tempo. De fato, seus nomes indicam a natureza desta variação. Suas transformadas também indicam como as frequências destes sinais variam com o tempo. Por exemplo, o chirp linear possui uma taxa de variação de sua frequência ao longo do tempo que é linear. Por sua vez, os componentes freqüenciais do chirp quadrático variam quadraticamente. O chirp gaussiano tem uma variação gaussiana da frequência, enquanto o chirp hiperbólico contém seus componentes freqüenciais variando conforme uma função hiperbólica.

Figura 1.3: Sinais chirp do Exercício 1 e seus respectivos espectros de potência para o domínio $t \in [0, 6]$.



Exercício 2

Neste exercício implementa-se a Transformada Janelada de Fourier (ou WFT, do inglês Windowed Fourier Trnasform) sobre os chirps do Exercício 1. Seis janelas são implementadas: Retangular, janela de Hanning, de Tukey, de Bartlett, de Papoulis e de Hamming. Elas estão ilustradas nas Figuras 2.1 e 2.3.

2.a

As janelas deste exercício foram assim definidas (gráficos das Figura 2.1 e 2.3):

Janela **retangular**:

```
1 def rect(window_size, loc, array_size, sr):
2     #-----
3     # Inputs:
4     # (float) window_size: size of window in physical units
5     # (int) loc: index to locate center of window in output array
6     # (int) array_size: size of output array
7     # (float) sr: sampling rate; used to scale window size from
8     #           physical
9     #           units to index values
10    # Outputs:
11    # (float array) r: array of zeros with a rectangular window
12    #           centered at loc
13    #-----
14    # Scaling window basde on sampling rate
15    window_size_SCALED = int((window_size)/sr)
16    # Return array
17    r = np.zeros(array_size)
18    # Time array (input of window function)
19    t = np.linspace(-1/2., 1/2., window_size_SCALED)
20    # Window function array
21    w = np.zeros(window_size_SCALED)
22    # Calculating window function values
23    w[np.where(abs(t) <= window_size_SCALED)] = 1
24    # Factor to match size of window location inside r and
25    # window_size
26    even_odd_w = window_size_SCALED%2
27    # If center of window too close to initial border
28    if loc < window_size_SCALED/2.:
29        # Truncating window
30        w = w[int(window_size_SCALED/2 -loc):]
```

```

31 # If center of window too close to final border
32 elif loc > array_size - window_size_SCALED / 2.:
33     # Truncating window
34     w = w[0:int(array_size - (loc - window_size_SCALED / 2.))]
35     # Creating aux. variables to calculate index location
36     r_init = int(loc - window_size_SCALED / 2.)
37     test = abs(array_size - r_init)
38     if test != len(w):
39         # Fixing index
40         r_init = array_size - len(w)
41     # Changing r values
42     r[r_init:] = w
43     # If center of window is completely inside array r
44 else:
45     # Changing r values
46     r[(loc - int(window_size_SCALED / 2.)):(loc + int(
47         window_size_SCALED / 2.) + even_odd_w)] = w
48     #
49 return r

```

Janela de **Hanning**:

```

1 def hanning(window_size, loc, array_size, sr):
2     window_size_SCALED = int(window_size/sr)
3     r = np.zeros(array_size)
4     t = np.linspace(-1/2., 1/2., window_size_SCALED)
5     w = 1/2. + (1/2.) * np.cos(2 * pi * t)
6     even_odd_w = window_size_SCALED%2
7     if loc < window_size_SCALED/2.:
8         w = w[int(window_size_SCALED/2 - loc):]
9         r[0:(len(w))] = w
10    elif loc > array_size - window_size_SCALED / 2.:
11        w = w[0:int(array_size - (loc - window_size_SCALED / 2.))]
12        r_init = int(loc - window_size_SCALED / 2.)
13        test = abs(array_size - r_init)
14        if test != len(w):
15            r_init = array_size - len(w)
16            r[r_init:] = w
17    else:
18        r[(loc - int(window_size_SCALED / 2.)):(loc + int(
19            window_size_SCALED / 2.) + even_odd_w)] = w
20
21 return r

```

Janela de **Hamming**:

```

1 def hamming(window_size, loc, array_size, sr):

```

```

2     window_size_SCALED = int(window_size/sr)
3     r = np.zeros(array_size)
4     t = np.linspace(-1/2., 1/2., window_size_SCALED)
5     w = .54 + .46 * np.cos(2 * pi * t)
6     even_odd_w = window_size_SCALED%2
7     if loc < window_size_SCALED/2.:
8         w = w[int(window_size_SCALED/2 -loc):]
9         r[0:(len(w))] = w
10    elif loc > array_size-window_size_SCALED/2.:
11        w = w[0:int(array_size-(loc-window_size_SCALED/2.))]
12        r_init = int(loc-window_size_SCALED/2.)
13        test = abs(array_size-r_init)
14        if test != len(w):
15            r_init = array_size - len(w)
16            r[r_init:] = w
17    else:
18        r[(loc-int(window_size_SCALED/2.)):(loc+int(
19        window_size_SCALED/2.)+even_odd_w)] = w
      return r

```

Janela de Bartlett:

```

1 def bartlett(window_size, loc, array_size, sr):
2     window_size_SCALED = int(window_size/sr)
3     r = np.zeros(array_size)
4     t = np.linspace(-1/2., 1/2., window_size_SCALED)
5     w = 1 - 2 * np.abs(t[np.where(abs(t) <= window_size_SCALED)])
6     even_odd_w = window_size_SCALED%2
7     if loc < window_size_SCALED/2.:
8         w = w[int(window_size_SCALED/2 -loc):]
9         r[0:(len(w))] = w
10    elif loc > array_size-window_size_SCALED/2.:
11        w = w[0:int(array_size-(loc-window_size_SCALED/2.))]
12        r_init = int(loc-window_size_SCALED/2.)
13        test = abs(array_size-r_init)
14        if test != len(w):
15            r_init = array_size - len(w)
16            r[r_init:] = w
17    else:
18        r[(loc-int(window_size_SCALED/2.)):(loc+int(
19        window_size_SCALED/2.)+even_odd_w)] = w
      return r

```

Janela de Papoulis:

```

1 def papoulis(window_size, loc, array_size, sr):

```

```

2     window_size_SCALED = int(window_size/sr)
3     r = np.zeros(array_size)
4     t = np.linspace(-1/2., 1/2., window_size_SCALED)
5     w = (1/pi) * np.abs(np.sin(2*pi*t[np.where(abs(t) <=
6         window_size_SCALED)])) + \
7             (1 - 2 * np.abs(t[np.where(
8             abs(t) <= window_size_SCALED)])) * \
9                 np.cos(2*pi*t[np.where(abs(
10            t) <= window_size_SCALED)])
11    even_odd_w = window_size_SCALED%2
12    if loc < window_size_SCALED/2.:
13        w = w[int(window_size_SCALED/2 -loc):]
14        r[0:len(w)] = w
15    elif loc > array_size - window_size_SCALED/2.:
16        w = w[0:int(array_size-(loc - window_size_SCALED/2.))]
17        r_init = int(loc - window_size_SCALED/2.)
18        test = abs(array_size - r_init)
19        if test != len(w):
20            r_init = array_size - len(w)
21            r[r_init:] = w
22        else:
23            r[(loc - int(window_size_SCALED/2.)): (loc + int(
24                window_size_SCALED/2.) + even_odd_w)] = w
25    return r

```

Janela de Tukey:

```

1 def tukeywin(window_length, alpha, loc, array_size, sr):
2     '''The Tukey window, also known as the tapered cosine window,
3     can be regarded as a cosine lobe of width \alpha * N / 2
4     that is convolved with a rectangle window of width (1 - \alpha
5     / 2). At \alpha = 1 it becomes rectangular, and
6     at \alpha = 0 it becomes a Hann window.
7
8     We use the same reference as MATLAB to provide the same results
9     in case users compare a MATLAB output to this function
10    output
11
12    Reference
13    -----
14    http://www.mathworks.com/access/helpdesk/help/toolbox/signal/
15    tukeywin.html
16    '',
17    window_length_SCALED = int(window_length/sr)
18    r = np.zeros(array_size)

```

```

15 # Special cases
16 if alpha <= 0:
17     return np.ones(window_length_SCALED) #rectangular window
18 elif alpha >= 1:
19     return np.hanning(window_length_SCALED)
20 # Normal case
21 x = np.linspace(0, 1, window_length_SCALED)
22 w = np.ones(x.shape)
23 # first condition 0 <= x < alpha/2
24 first_condition = x<alpha/2
25 w[first_condition] = 0.5 * (1 + np.cos(2*np.pi/alpha * (x[
26 first_condition] - alpha/2)))
27 # second condition already taken care of
28 # third condition 1 - alpha / 2 <= x <= 1
29 third_condition = x>=(1 - alpha/2)
30 w[third_condition] = 0.5 * (1 + np.cos(2*np.pi/alpha * (x[
31 third_condition] - 1 + alpha/2)))
32 even_odd_w = window_length_SCALED%2
33 if loc < window_length_SCALED/2.:
34     w = w[int(window_length_SCALED/2 -loc):]
35     r[0:(len(w))] = w
36 elif loc > array_size-window_length_SCALED/2.:
37     w = w[0:int(array_size-(loc-window_length_SCALED/2.))]
38     r_init = int(loc-window_length_SCALED/2.)
39     test = abs(array_size-r_init)
40     if test != len(w):
41         r_init = array_size - len(w)
42     r[r_init:] = w
43 else:
44     r[(loc-int(window_length_SCALED/2.)): (loc+int(
45 window_length_SCALED/2.))+even_odd_w)] = w
46 return r

```

As funções janela estão graficadas com tamanho igual a 0.5 sobre valores de t de -0.4 a +0.4 na Figura 2.1. Suas transformadas estão na Figura 2.2.

As mesmas janelas, porém com tamanho igual a 0.1, produzem os resultados das Figuras 2.3 e 2.4.

Figura 2.1: Gráfico das seis janelas utilizadas neste exercício com largura (abaixo denominado `window size`) igual a 0.5.

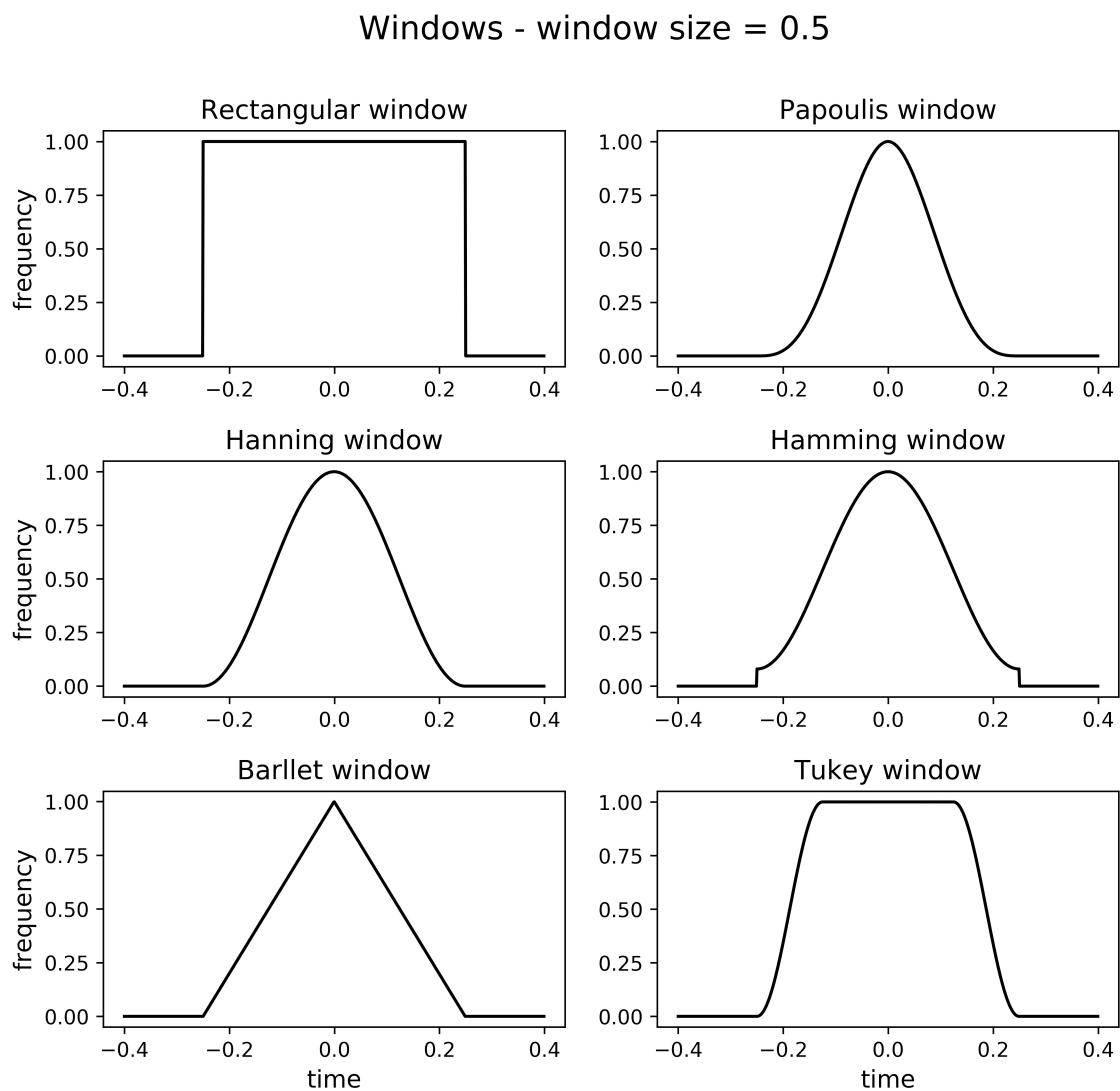


Figura 2.2: Transformadas de Fourier das funções janela utilizadas neste exercício com largura igual a 0.5.

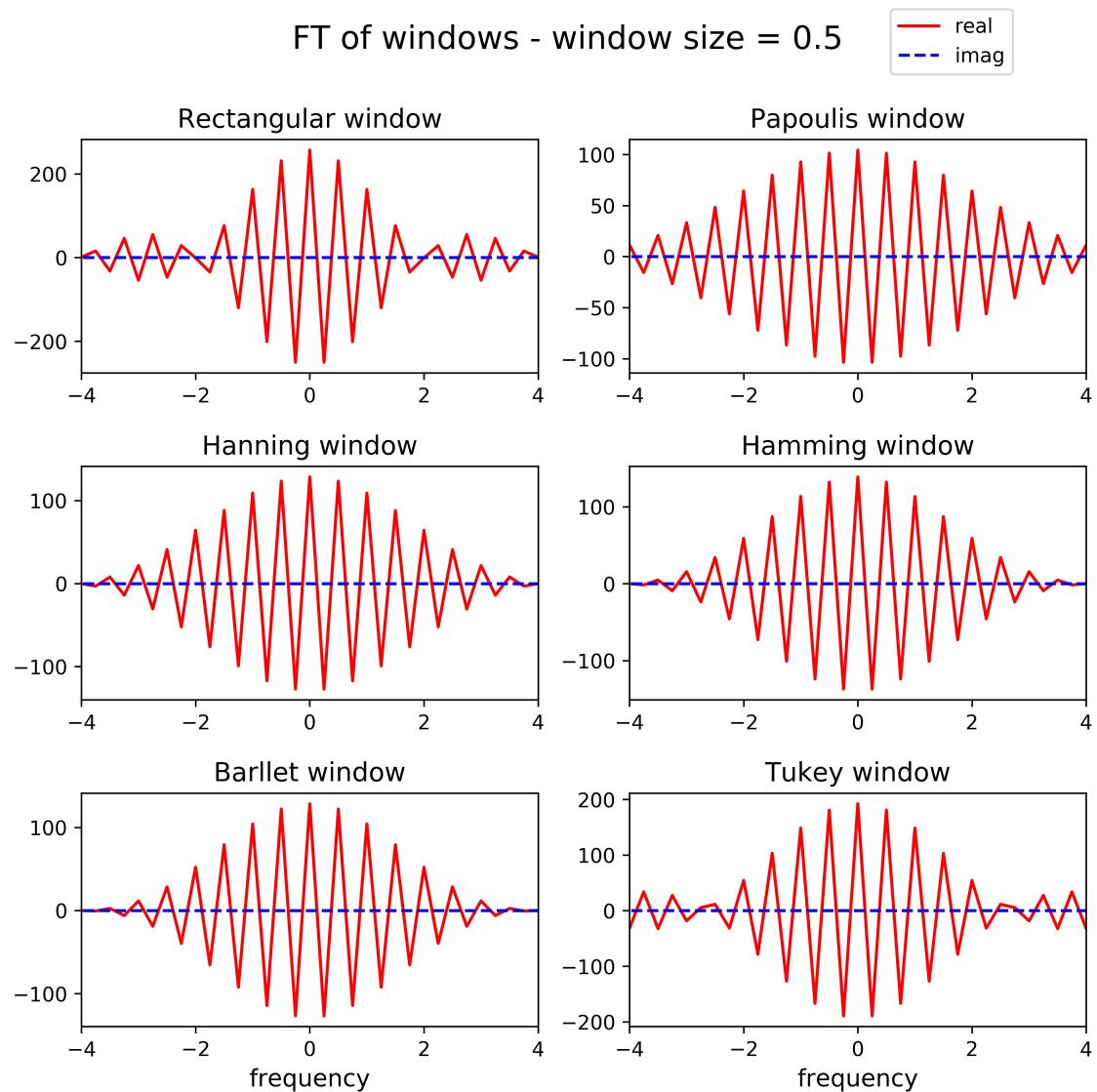


Figura 2.3: Gráfico das seis janelas utilizadas neste exercício com largura (abaixo denominado `window size`) igual a 0.1.

Windows - window size = 0.1

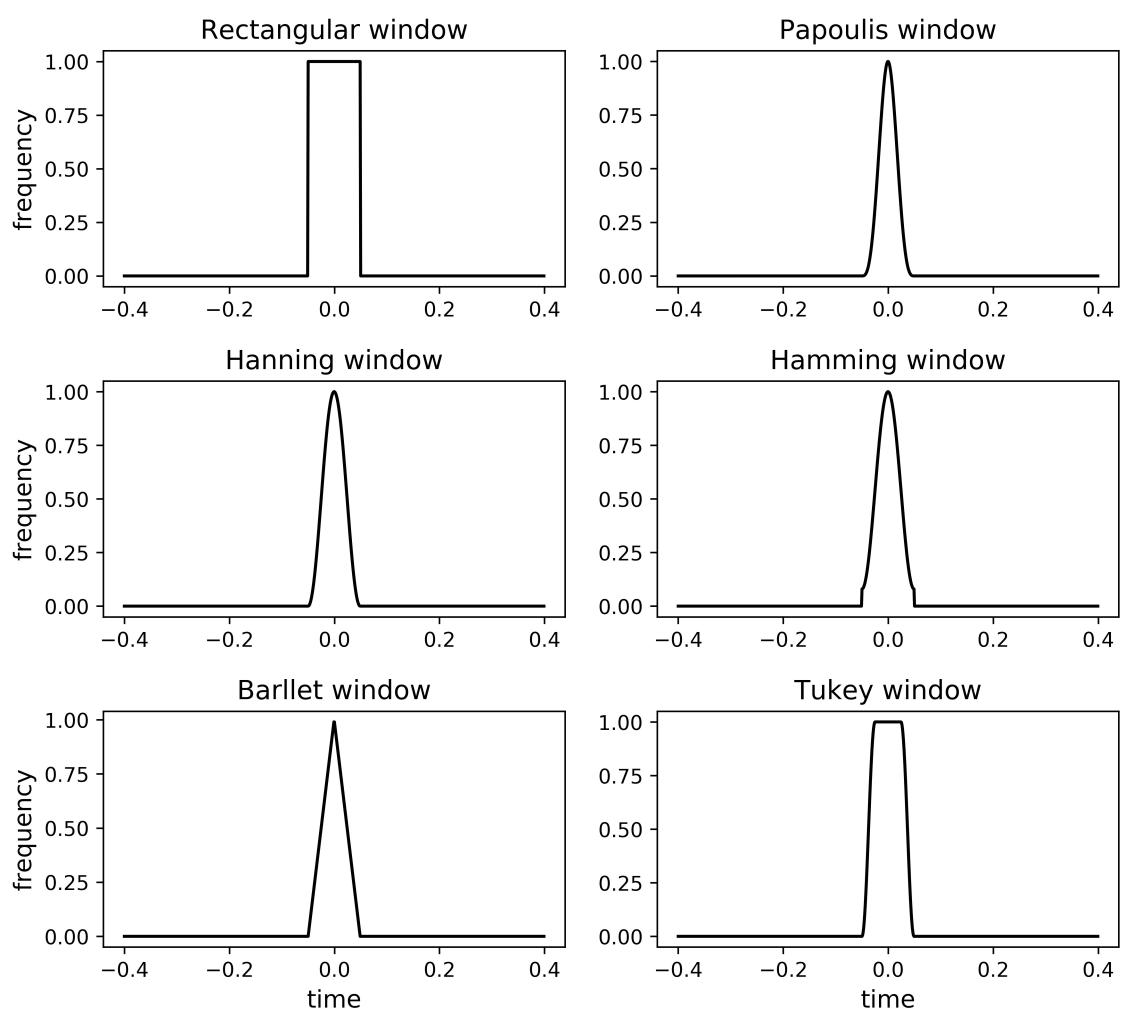
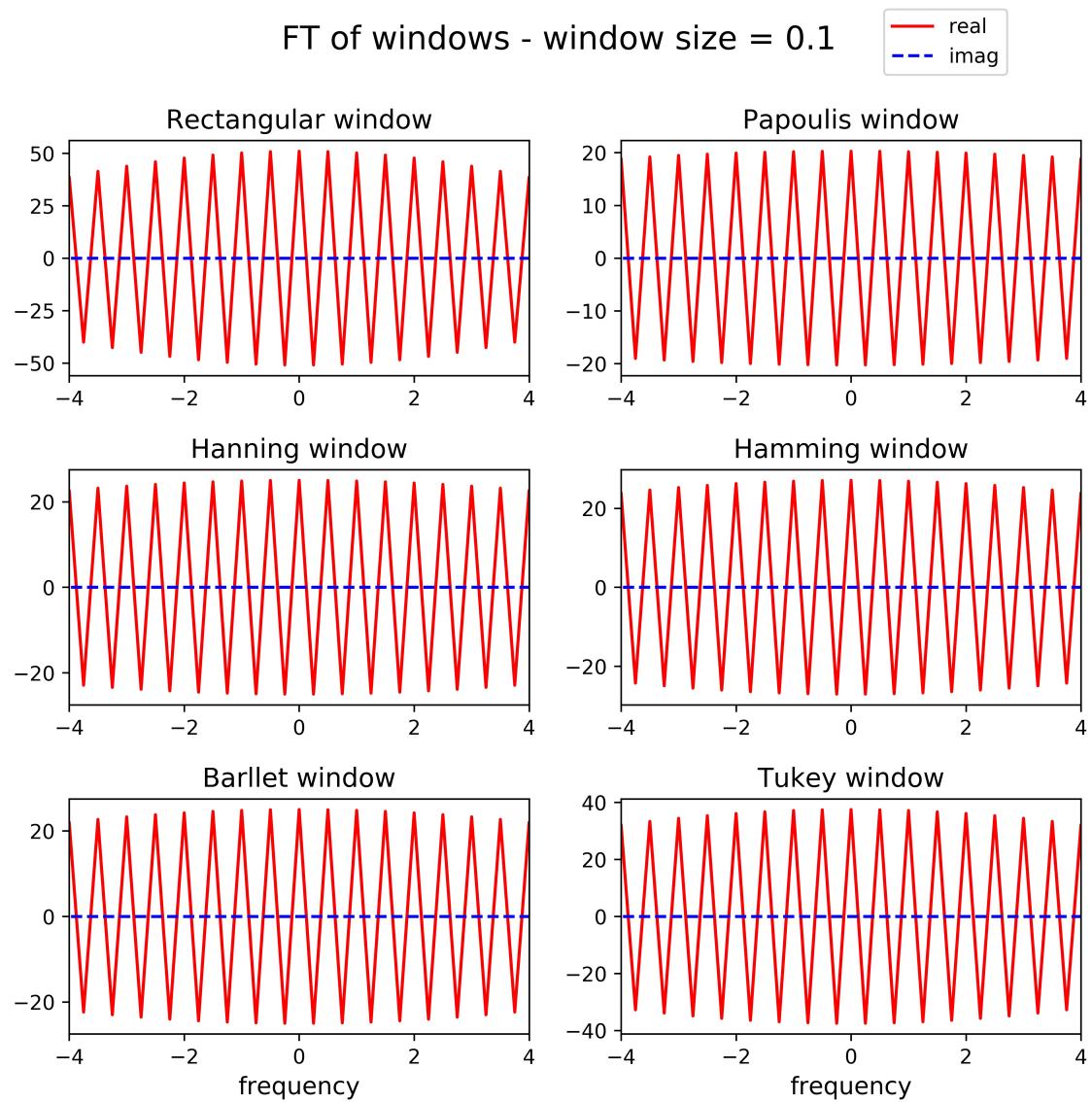


Figura 2.4: Transformadas de Fourier das funções janela utilizadas neste exercício com largura igual a 0.1.



2.b

Dada uma escolha do sinal (variável `chirp` abaixo, à qual é atribuída um dos chirps do Exercício 1), os espectrogramas foram calculados a partir do método `spec`, recebendo como input: o produto do chirp e da função janela *shiftada*, a resolução `res` (para cálculo dos bins de frequência), a frequência final de interesse `f_final`, e o número de bins `N` com o qual a FFT será implementada.

```
1 # Defining spectrogram arrays
2 spec_1 = np.zeros([f_final, len(tau)])
3 spec_2 = np.zeros([f_final, len(tau)])
4 spec_3 = np.zeros([f_final, len(tau)])
5 spec_4 = np.zeros([f_final, len(tau)])
6 spec_5 = np.zeros([f_final, len(tau)])
7 spec_6 = np.zeros([f_final, len(tau)])
8 # Creating spectrograms
9 for i in range(len(chirp)):
10     # Printing progress
11     print(str(round(100*i/len(chirp),2))+'%')
12     # Center of window
13     loc = i
14     #----- rectangular
15     # Creating window array
16     w1 = rect(window_size, loc, array_size, res)
17     # Windowed spectra centered at loc
18     x1, y1 = spec(chirp * w1, res, f_final, N)
19     # Updating spectrogram array
20     spec_1[:,i]=y1
21     #----- hanning
22     w2 = hanning(window_size, loc, array_size, res)
23     x2, y2 = spec(chirp * w2, res, f_final, N)
24     spec_2[:,i]=y2
25     #----- bartlett
26     w3 = bartlett(window_size, loc, array_size, res)
27     x3, y3 = spec(chirp * w3, res, f_final, N)
28     spec_3[:,i]=y3
29     #----- papoulis
30     w4 = papoulis(window_size, loc, array_size, res)
31     x4, y4 = spec(chirp * w4, res, f_final, N)
32     spec_4[:,i]=y4
33     #----- hamming
34     w5 = hamming(window_size, loc, array_size, res)
35     x5, y5 = spec(chirp * w5, res, f_final, N)
36     spec_5[:,i]=y5
37     #----- tukey
```

```

38     w6 = tukeywin(window_size, 0.5, loc, array_size, res)
39     x6, y6 = spec(chirp * w6, res, f_final, N)
40     spec_6[:,i]=y6
41 # Deleting garbage
42 del w1, w2, w3, w4, w5, w6
43 del y1, y2, y3, y4, y5, y6
44 del x2, x3, x4, x5, x6
45 # Spectrograms for plot
46 spec_1_p = 20*np.log10(np.abs(spec_1))
47 spec_2_p = 20*np.log10(np.abs(spec_2))
48 spec_3_p = 20*np.log10(np.abs(spec_3))
49 spec_4_p = 20*np.log10(np.abs(spec_4))
50 spec_5_p = 20*np.log10(np.abs(spec_5))
51 spec_6_p = 20*np.log10(np.abs(spec_6))

```

A seguir são apresentados os resultados das WFT de cada chirp do Exercício 1. Dois valores foram usados para o tamanho das janelas implementadas e dois domínios diferentes para cada chirp (a saber, os domínios referentes às Figuras 1.1 e 1.2). O espectrograma de cada caso foi gerado.

As figuras dos espectrogramas foram criadas com o pacote `matplotlib`, em particular com a função `contourf`. Foi utilizando o mapa de cor `jet` e 256 níveis de cor. Cada figura possui o mesmo intervalo de cor em sua representação (mesmos `vmin` e `vmax`). O trecho do código que gera a visualização das Figuras 2.4 a 2.16 é exibido abaixo:

```

1 fig, ax = plt.subplots(3, 2, figsize=(10,8)) # (3 x 2) subplots
2 # Global settings
3 # colormap
4 my_cmap = matplotlib.cm.get_cmap('jet')
5 fig.suptitle(plot_title, size=16)
6 # x ticks and labels
7 t_ticks = np.linspace(0, len(t)-1, xamount, dtype=int)
8 t_labels = list(np.round(t[t_ticks], yround))
9 t_labels[t_labels.index(0)]=0
10 # y ticks and labels
11 f_ticks = np.linspace(0, len(x1)-1, yamount, dtype=int)
12 f_labels = list(np.round(x1[f_ticks], yround))
13 # minimum and maximum values for color value in all plots
14 p_min = np.min([spec_1_p, spec_2_p, spec_3_p, spec_4_p, spec_5_p])
15 p_max = np.max([spec_1_p, spec_2_p, spec_3_p, spec_4_p, spec_5_p])
16 #----- Row 1 Col 1
17 ax[0,0].set_title('Rectangular window', size = 13)
18 ax[0,0].contourf(spec_1_p, 256, origin='lower', cmap=my_cmap, vmin=

```

```

    p_min, vmax=p_max)
19 ax[0,0].set_ylabel('frequency', size=12)
20 #----- Row 2 Col 1
21 ax[1,0].set_title('Hanning window', size=13)
22 ax[1,0].contourf(spec_2_p, 256, origin='lower', cmap=my_cmap, vmin=
    p_min, vmax=p_max)
23 ax[1,0].set_ylabel('frequency', size=12)
24 #----- Row 3 Col 1
25 ax[2,0].set_title('Bartlett window', size=13)
26 ax[2,0].contourf(spec_3_p, 256, origin='lower', cmap=my_cmap, vmin=
    p_min, vmax=p_max)
27 ax[2,0].set_ylabel('frequency', size=12)
28 ax[2,0].set_xlabel('time', size=12)
29 #----- Row 1 Col 2
30 ax[0,1].set_title('Papoulis window', size=13)
31 im = ax[0,1].contourf(spec_4_p, 256, origin='lower', cmap=my_cmap,
    vmin=p_min, vmax=p_max)
32 #----- Row 1 Col 2
33 ax[1,1].set_title('Hamming window', size=13)
34 ax[1,1].contourf(spec_5_p, 256, origin='lower', cmap=my_cmap, vmin=
    p_min, vmax=p_max)
35 #----- Row 1 Col 3
36 ax[2,1].set_title('Tukey window', size=13)
37 ax[2,1].contourf(spec_6_p, 256, origin='lower', cmap=my_cmap, vmin=
    p_min, vmax=p_max)
38 ax[2,1].set_xlabel('time', size=12)
39 #----- End of individual-plot settings
40 for i in range(3):
    # Setting same x ticks and tick labels for all plots
41     ax[i,0].set_xticks(t_ticks)
42     ax[i,0].set_xticklabels(t_labels)
43     ax[i,1].set_xticks(t_ticks)
44     ax[i,1].set_xticklabels(t_labels)
45     #
46     ax[i,0].set_yticks(f_ticks)
47     ax[i,0].set_yticklabels(f_labels)
48     ax[i,1].set_yticks(f_ticks)
49     ax[i,1].set_yticklabels(f_labels)
50
51
52 # Setting spacing between plots
53 fig.subplots_adjust(left=None, bottom=None, right=.85, top=None,
    wspace=.2, hspace=.4)
54 # Setting colorbar
55 cbar_ax = fig.add_axes([0.88, 0.15, 0.03, 0.7])
56 cb = fig.colorbar(im, cax=cbar_ax)

```

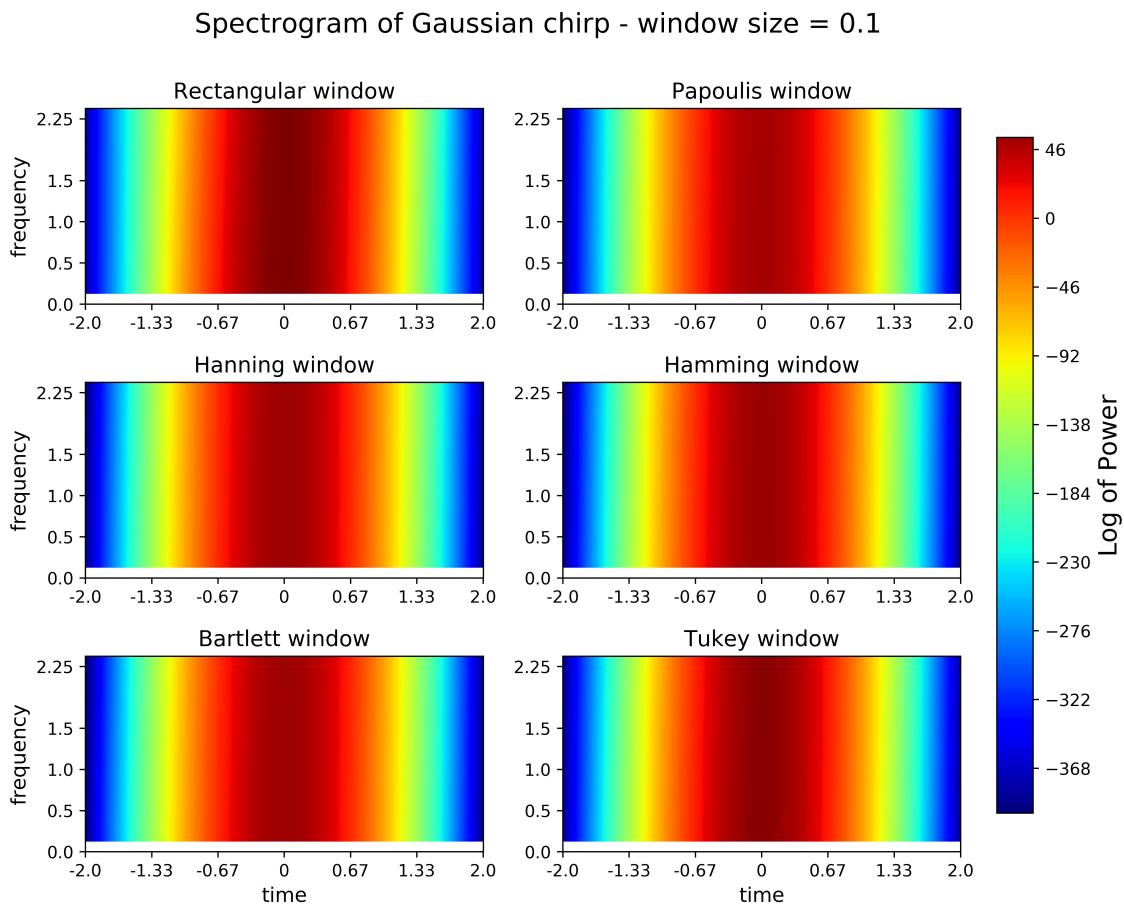
```

57 cb.set_label('Log of Power', fontsize=15)
58 #----- Saving and showing plot
59 fig.savefig(fig_name+'.jpg', dpi=600, bbox_inches='tight')
60 plt.show()

```

Espectrogramas do **chirp gaussiano**, $-2 \leq t \leq 2$ e tamanho das janelas = 0.1:
Figura 2.4.

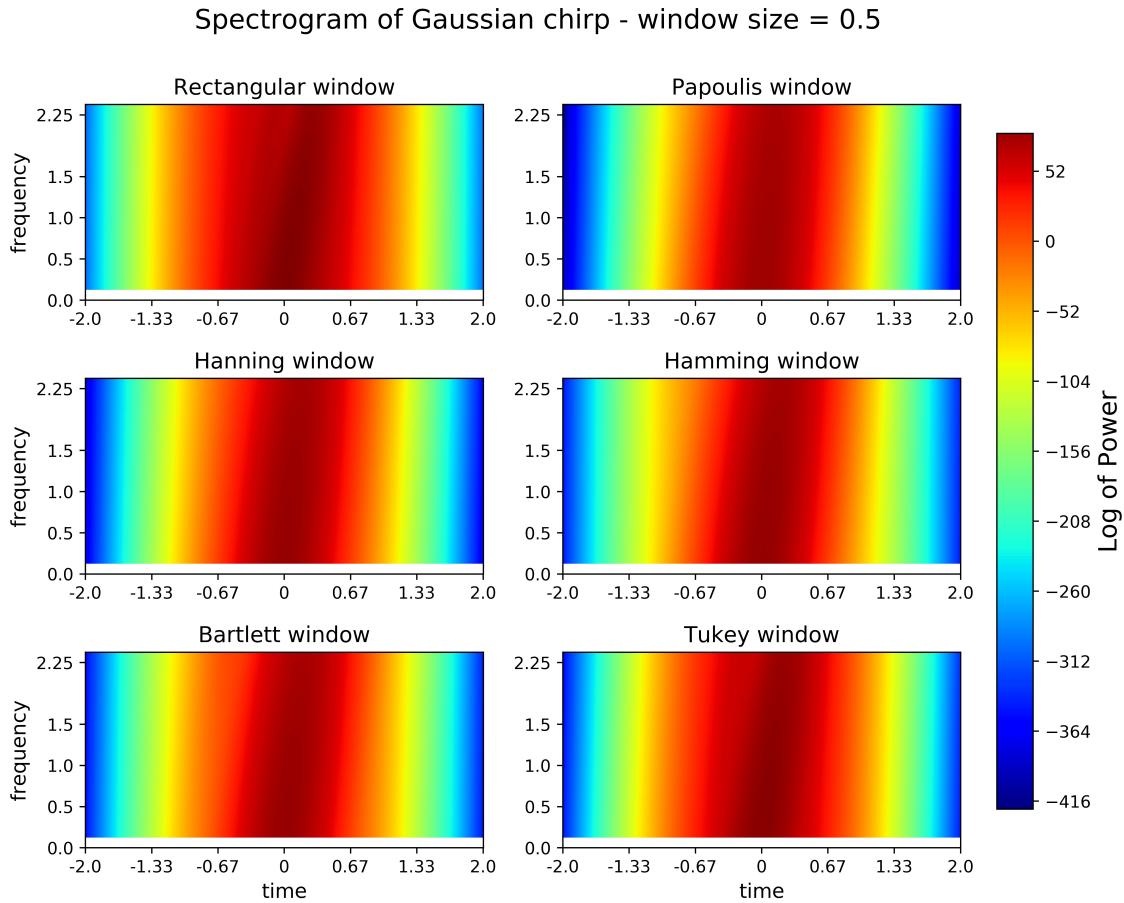
Figura 2.4: Espectrogramas do chirp gaussiano com as seis funções janela usadas neste exercício de tamanho (largura) igual a 0.1. Comparar com o topo da Figura 1.1.



Espectrogramas do **chirp gaussiano**, $-2 \leq t \leq 2$ e tamanho das janelas = 0.5:
Figura 2.5.

Espectrogramas do **chirp gaussiano**, $0 \leq t \leq 6$ e tamanho das janelas = 0.5: Figura 2.6.

Figura 2.5: Espectrogramas do chirp gaussiano com as seis funções janela usadas neste exercício de tamanho (largura) igual a 0.5. Comparar com o topo da Figura 1.1.



Espectrogramas do **chirp linear**, $-6 \leq t \leq 6$ e tamanho das janelas = 0.1: Figura 2.7.

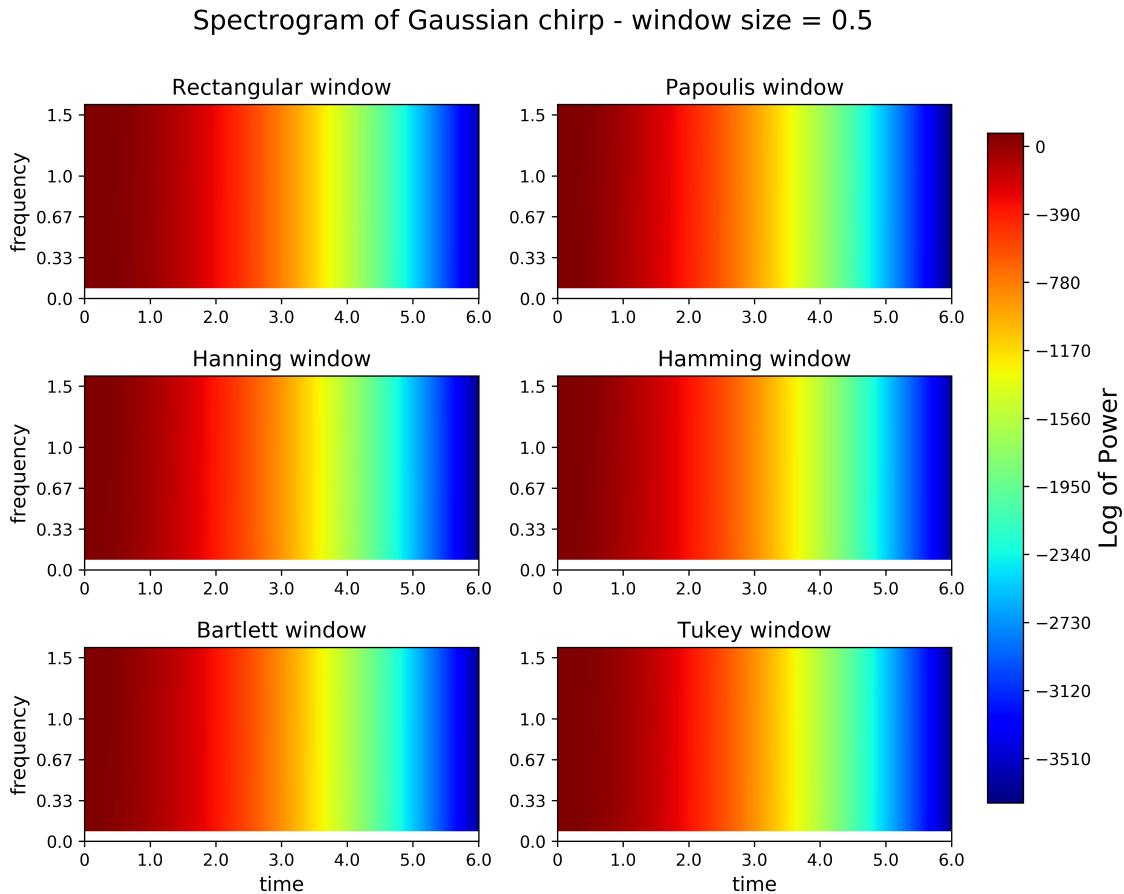
Espectrogramas do **chirp linear**, $-6 \leq t \leq 6$ e tamanho das janelas = 0.5: Figura 2.8.

Espectrogramas do **chirp linear**, $0 \leq t \leq 6$ e tamanho das janelas = 0.5: Figura 2.9.

Espectrogramas do **chirp quadrático**, $-4 \leq t \leq 4$ e tamanho das janelas = 0.1: Figura 2.10.

Espectrogramas do **chirp quadrático**, $-4 \leq t \leq 4$ e tamanho das janelas = 0.5: Figura 2.11.

Figura 2.6: Espectrogramas do chirp gaussiano com as seis funções janela usadas neste exercício de tamanho (largura) igual a 0.5. Comparar com o topo da Figura 1.3.



Espectrogramas do **chirp quadrático**, $0 \leq t \leq 6$ e tamanho das janelas = 0.5: Figura 2.12.

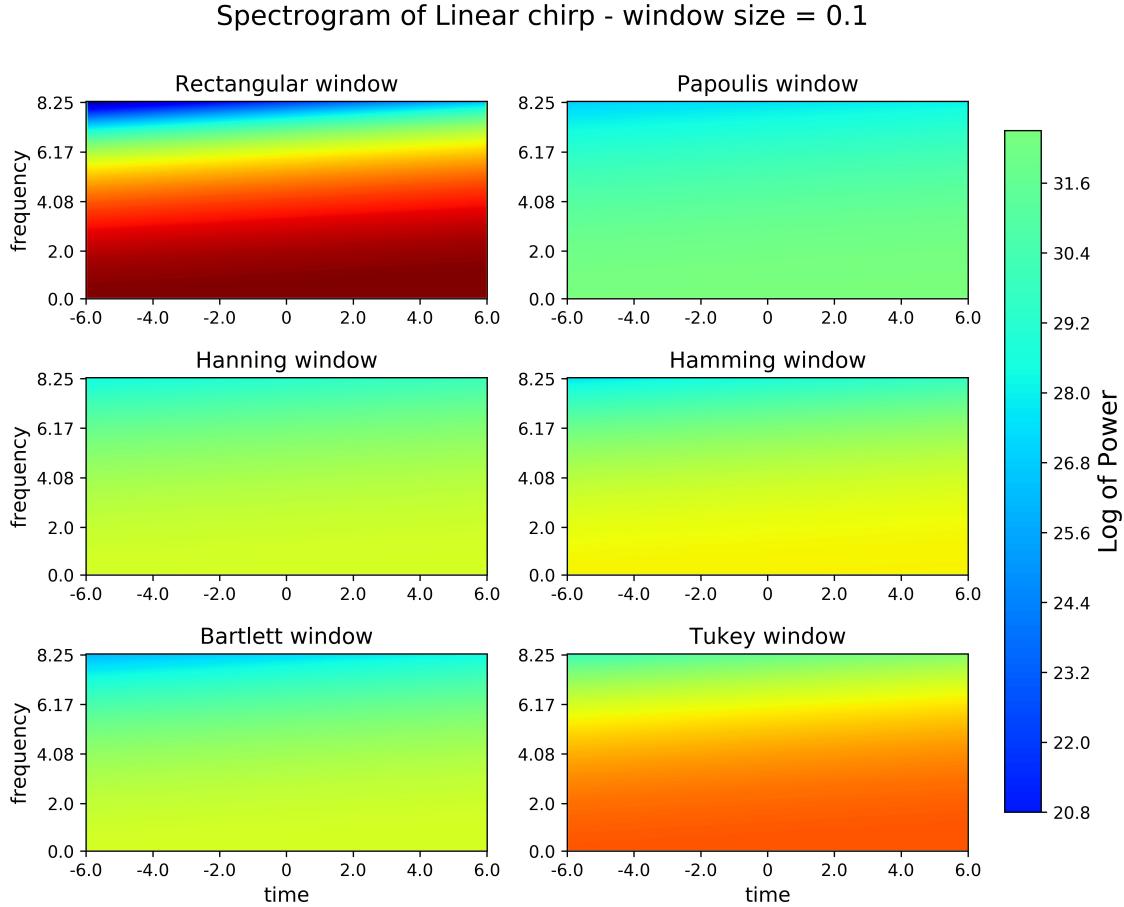
Espectrogramas do **chirp hiperbólico**, $0 \leq t \leq 2$ e tamanho das janelas = 0.1: Figura 2.13.

Espectrogramas do **chirp hiperbólico**, $0 \leq t \leq 2$ e tamanho das janelas = 0.5: Figura 2.14.

Espectrogramas do **chirp hiperbólico**, $0 \leq t \leq 6$ e tamanho das janelas = 0.1: Figura 2.15.

Espectrogramas do **chirp hiperbólico**, $0 \leq t \leq 6$ e tamanho das janelas = 0.5: Figura 2.16.

Figura 2.7: Espectrogramas do chirp linear com janelas de tamanho igual a 0.1. Comparar com segunda linha (de cima para baixo) da Figura 1.1.



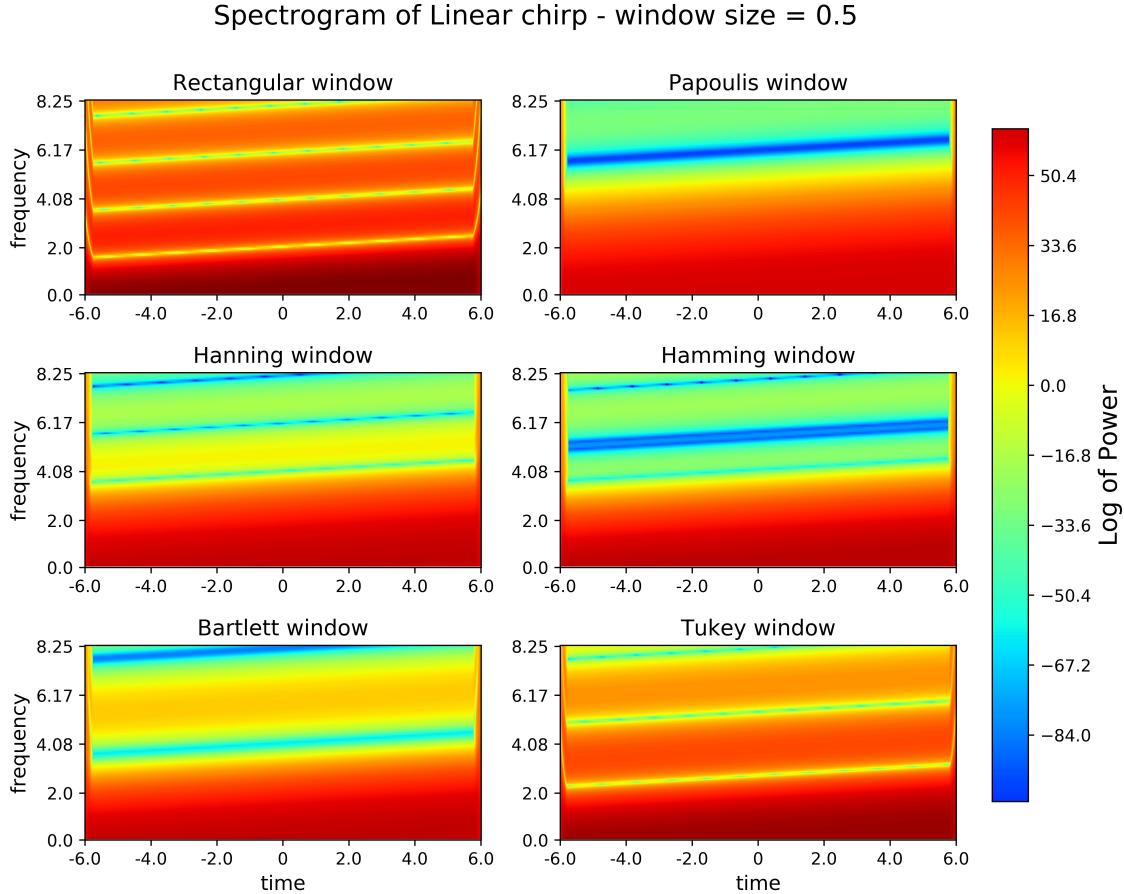
2.c

Se o objetivo da análise é obter detalhes da variação das diferentes frequências do sinal ao longo do tempo, a implementação das WFT com janelas grandes é desejada, uma vez que isso oferecerá maior resolução freqüencial no espectrograma. Para o chirp gaussiano, uma vez que a variação da frequência é suave e com poucos componentes, as janelas de tamanho 0.1 e 0.5 (Figuras 2.4, 2.5 e 2.6) foram equivalentes em resultado.

Já para o chirp linear, a janela de tamanho igual a 0.1 não foi capaz de captar a variação linear (e suave) da frequência (Figura 2.7). As janelas de tamanho 0.5 conseguiram.

As Figuras 2.10 e 2.11 (chirp quadrático), quando contrastadas, também ilustram a incapacidade das janelas de tamanho igual a 0.1 de contribuir para a análise deste

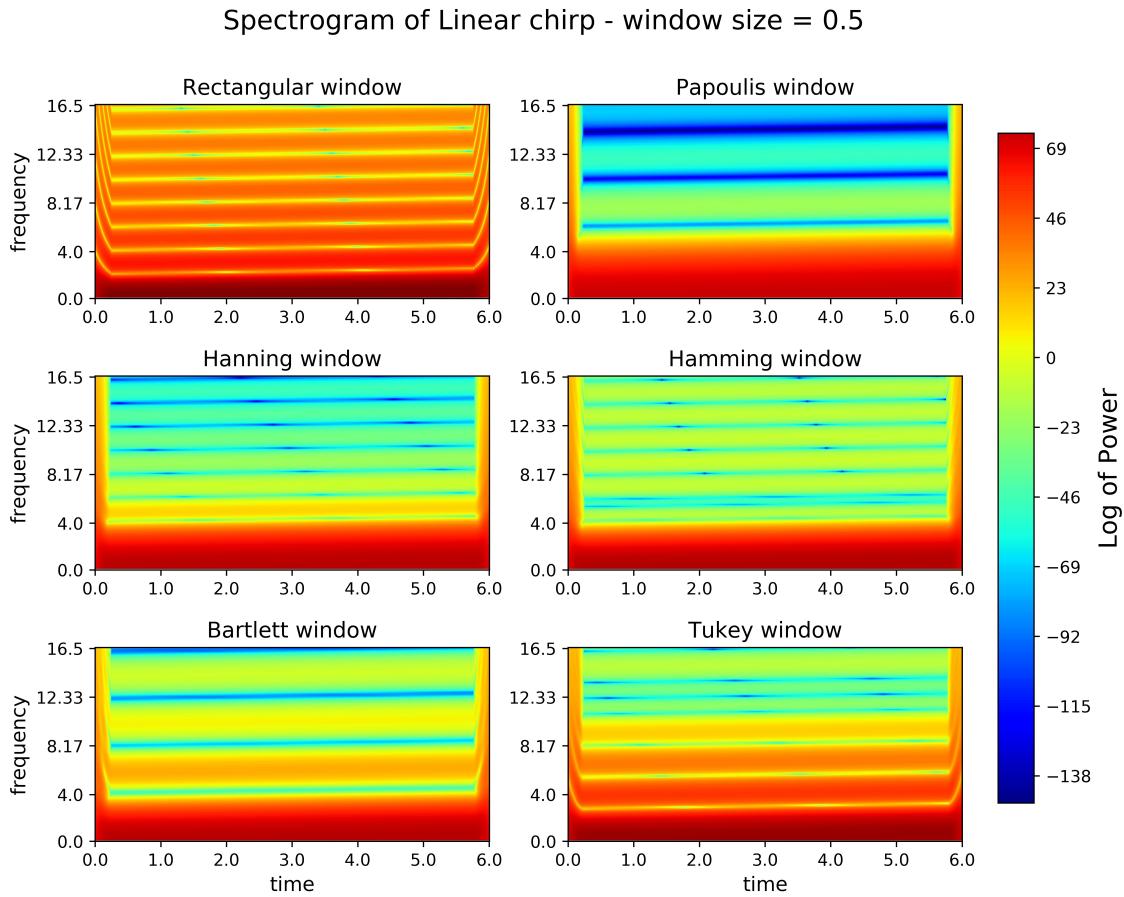
Figura 2.8: Espectrogramas do chirp linear com janelas de tamanho igual a 0.5. Comparar com segunda linha (de cima para baixo) da Figura 1.1.



chirp. Na Figura 2.10, assim como na Figura 2.7, somente a janela retangular foi capaz de captar uma variação apreciável de frequência. Isso pode ser explicado pela Figura 2.1, que ilustra a transformada da janela retangular como tendo a menor largura (em frequência) e, portanto, melhor resolução freqüencial.

O chirp hiperbólico foi o sinal que melhor ilustrou o *trade-off* da resolução tempo × frequência por trás da definição do tamanho da janela. As Figuras 2.13 e 2.15 exibem os espectrogramas para as janelas de tamanho igual a 0.1, ou seja, de alta resolução temporal. A variação das cores (das frequências) na direção do eixo horizontal nestas figuras é bem acentuada. A WFT com estas janelas captou bem a variação temporal do sinal sem determinar bem quais frequências compõem o sinal localmente. Em contrapartida, nas Figura 2.14 e 2.16 o tamanho das janelas é 0.5, ou seja, a resolução temporal é menor. Com isso, o espectrograma exibe forte variação de cores que resulta da melhor representação de frequências, ganhando resolução vertical ao passo que perde a capacidade de descrever a variação horizontal dessas cores como antes.

Figura 2.9: Espectrogramas do chirp linear com janelas de tamanho igual a 0.5. Comparar com segunda linha (de cima para baixo) da Figura 1.3.



Em conjunto, as figuras desta seção atestam a capacidade da WFT de analisar conteúdos freqüenciais de um sinal localmente, conferindo uma componente extra de análise: o tempo. Este componente é ausente na análise de Fourier tradicional, que possui um caráter global de análise. Com a WFT surgem parâmetros relevantes à nossa análise desde a análise de Fourier, a saber, o tamanho da função janela escolhida, diretamente relacionado ao já conhecido princípio da incerteza. Quanto maior (menor) a largura da janela, menor (maior) será a resolução temporal da ferramenta e maior (menor) será sua resolução freqüencial. Por fim, outro importante parâmetro da WFT é a função janela em si, uma vez que os resultados deste exercício sugerem que, em alguns casos, algumas funções janela são mais úteis que outras em captar as diferentes freqüências de um sinal.

Figura 2.10: Espectrogramas do chirp quadrático com janelas de tamanho igual a 0.1. Comparar com terceira linha (de cima para baixo) da Figura 1.1.

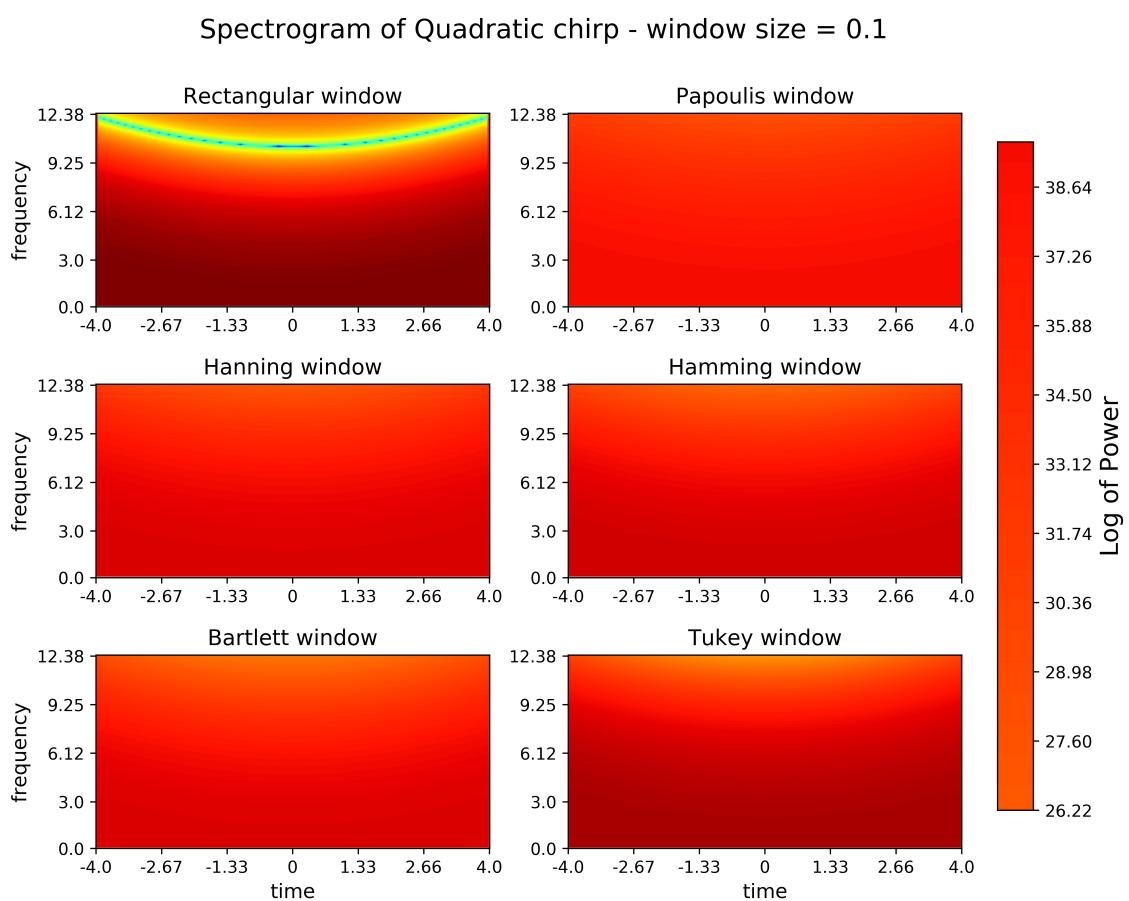


Figura 2.11: Espectrogramas do chirp quadrático com janelas de tamanho igual a 0.5. Comparar com terceira linha (de cima para baixo) da Figura 1.1.

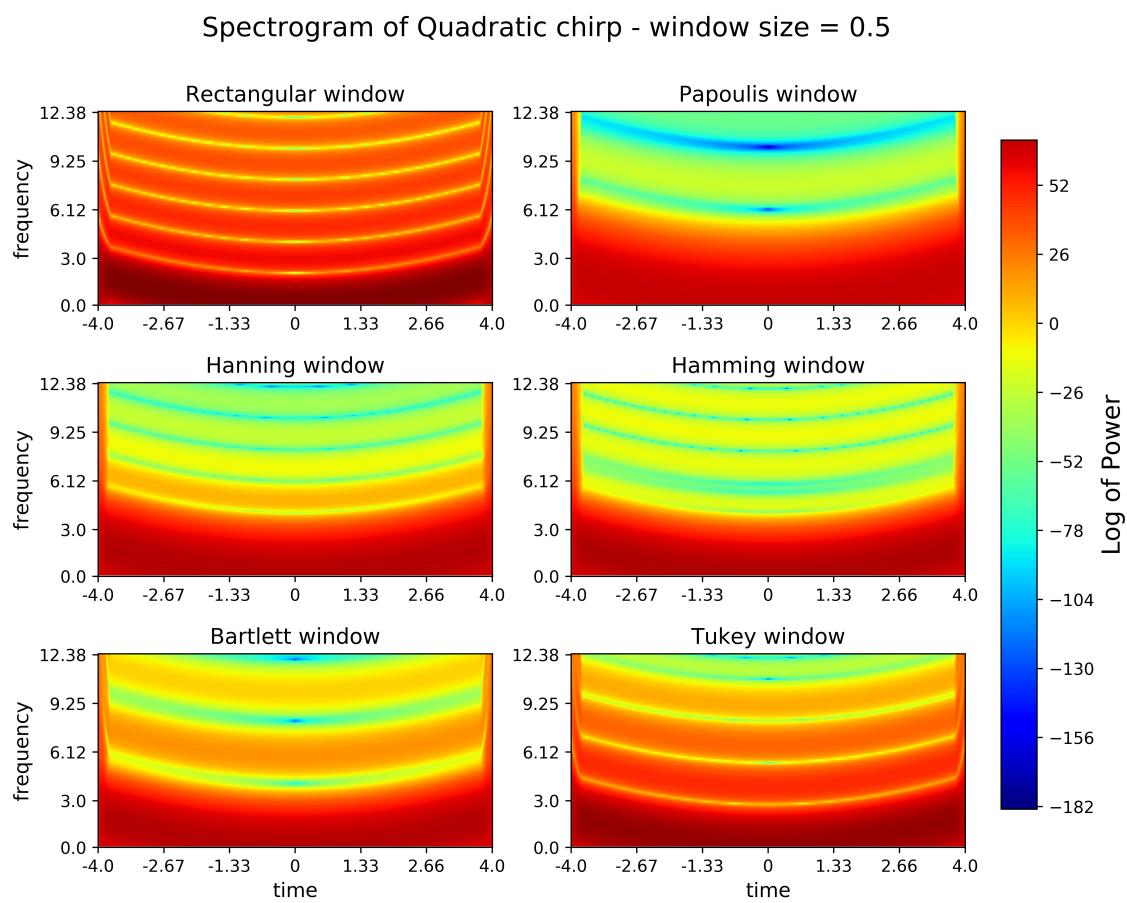


Figura 2.12: Espectrogramas do chirp quadrático com janelas de tamanho igual a 0.5. Comparar com terceira linha (de cima para baixo) da Figura 1.3.

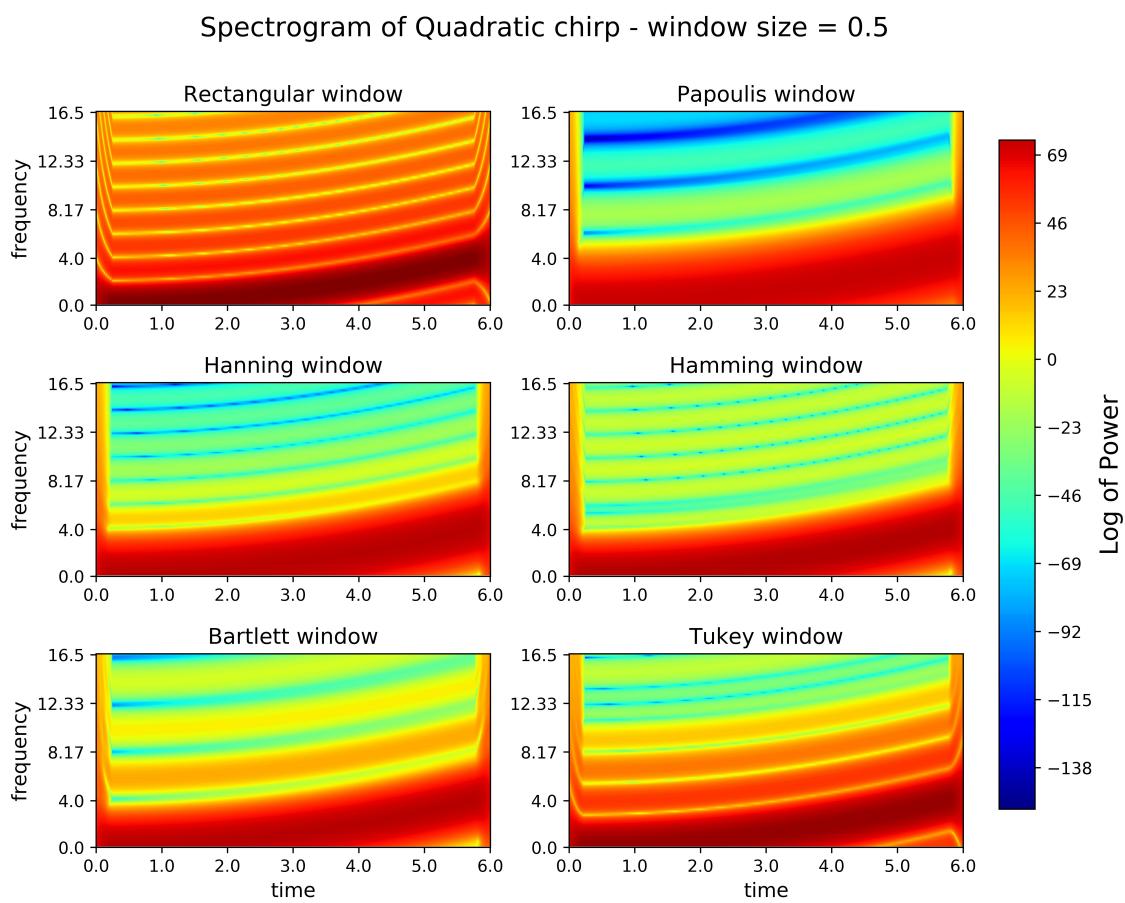


Figura 2.13: Espectrogramas do chirp hiperbólico com janelas de tamanho igual a 0.1. Comparar com última linha da Figura 1.1.

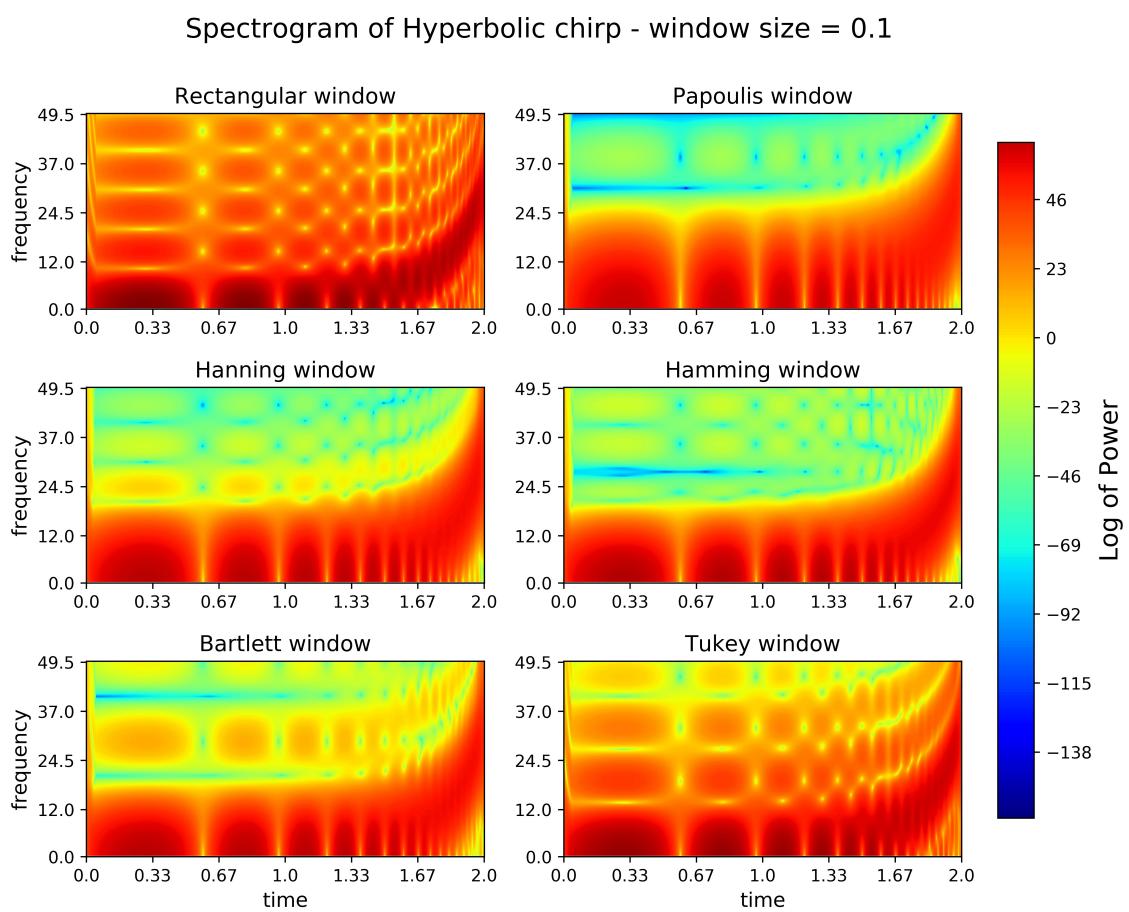


Figura 2.14: Espectrogramas do chirp hiperbólico com janelas de tamanho igual a 0.5. Comparar com última linha da Figura 1.1.

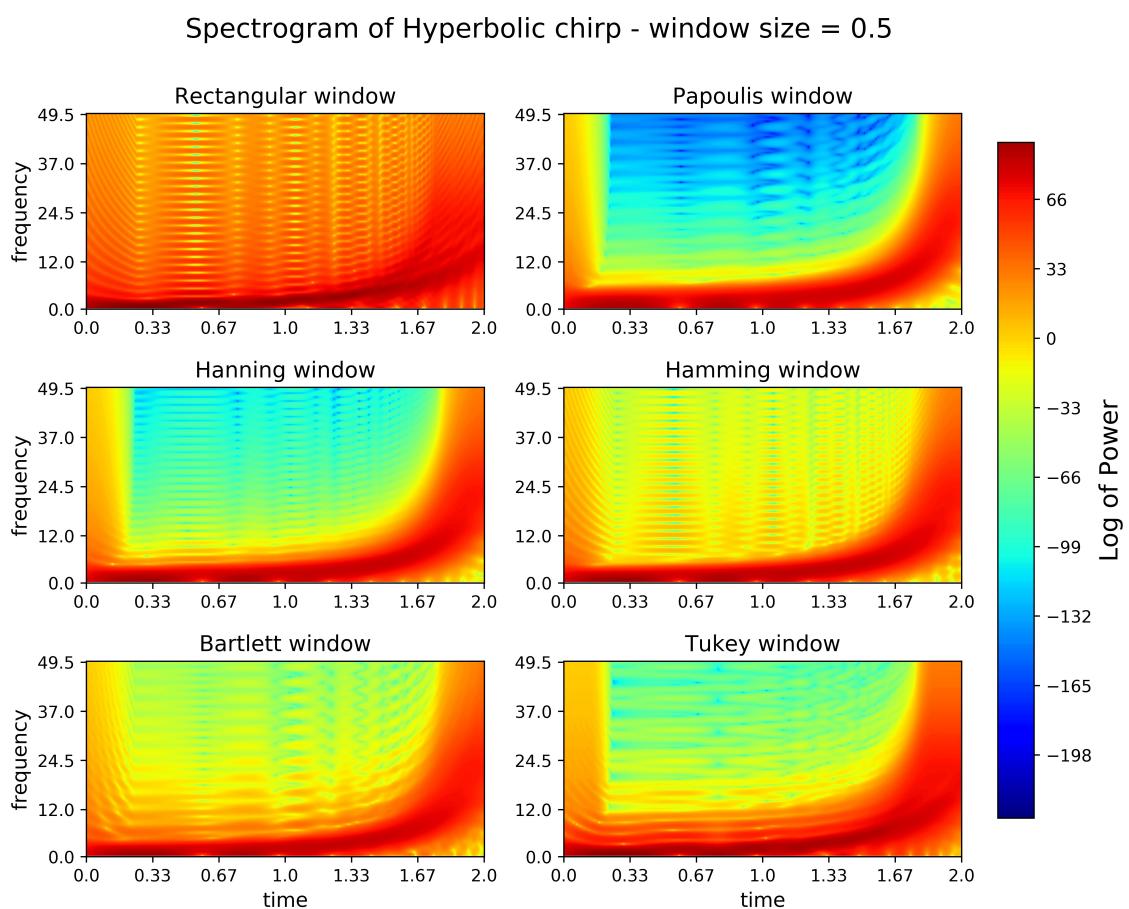


Figura 2.15: Espectrogramas do chirp hiperbólico com janelas de tamanho igual a 0.1. Comparar com última linha da Figura 1.3.

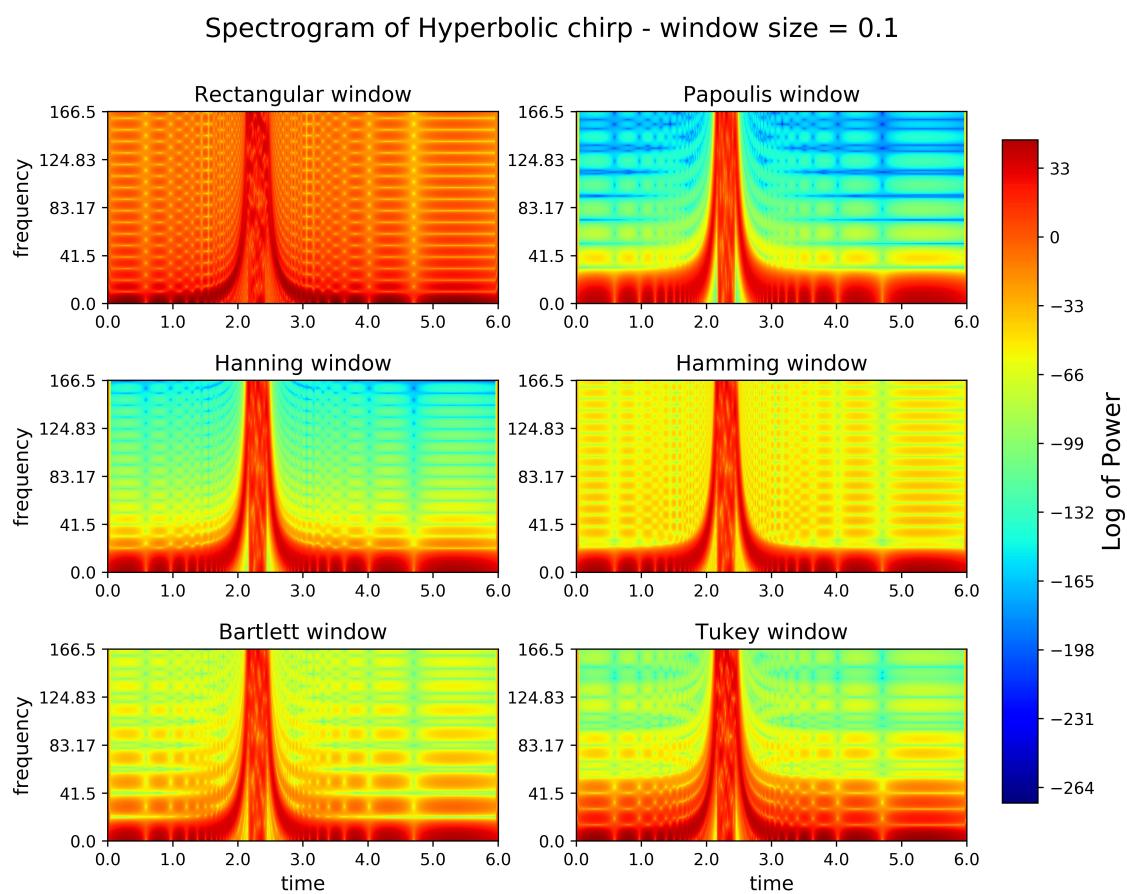
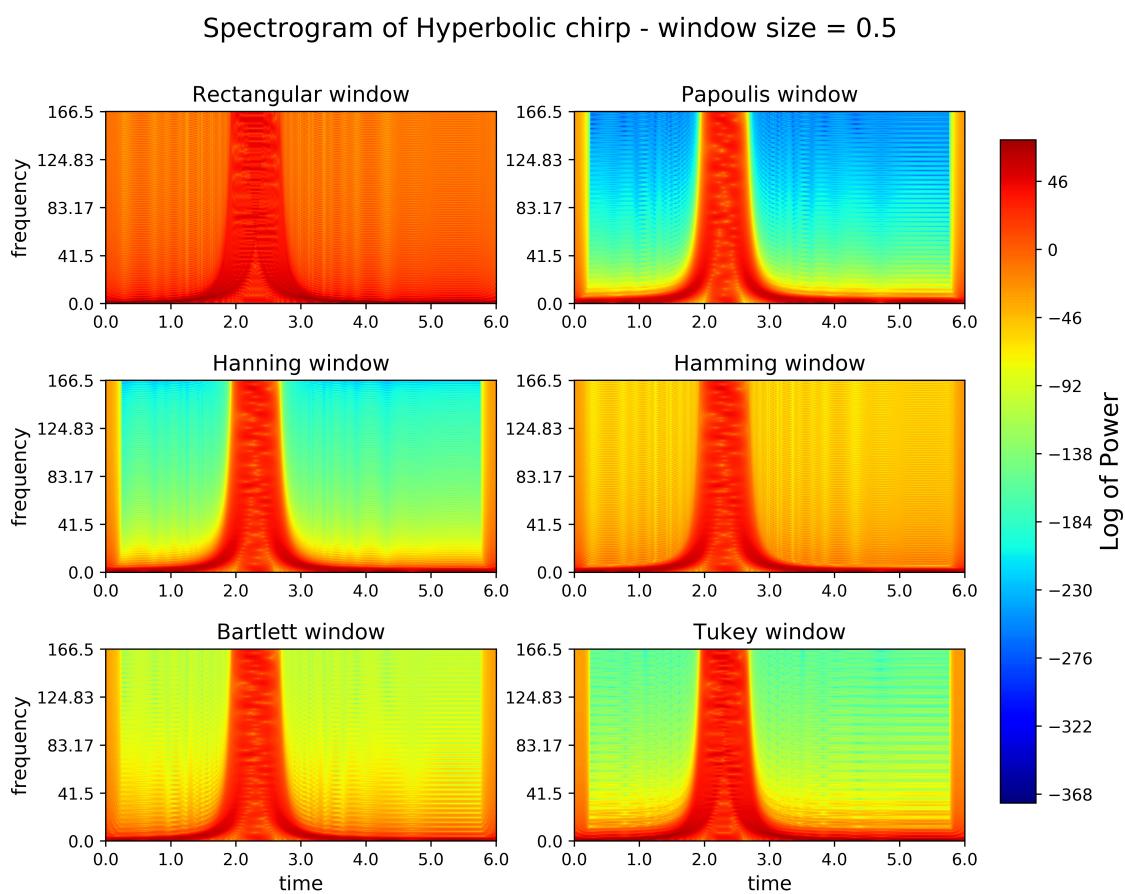


Figura 2.16: Espectrogramas do chirp hiperbólico com janelas de tamanho igual a 0.5. Comparar com última linha da Figura 1.3.



Exercício 3

3.1

Resultados presentes no relatório.

3.2

Relatório entregue separadamente.

Exercício 4

4.1

Resultados presentes no relatório.

4.2

Resultados presentes no relatório.

4.3

Resultados presentes no relatório.

4.4

Resultados presentes no relatório.

4.5

Relatório em preparação.