Enviado: 01/09/2021 *Recibido: 10/09/2021* *Aceptado: 06/01/2022*

UNIVERSIDAD DISTRITAL FRANCISCO JOSE DE CALDAS

VISIÓN ELECTRÓNICA

Algo más que un estado sólido

https://doi.org/10.14483/issn.2248-4728

VISIÓN ELECTRÓNICA

A RESEARCH VISION

Functions and codes for obtaining the normalized power spectral density with free software

Funciones y códigos para obtener la densidad espectral de potencia normalizada con software libre

Gerardo Castang-Montiel[1], Rocío Rodríguez-Guerrero[2], Carlos Alberto Vanegas[3]

Abstract

In this paper, the codes implemented for obtaining the power spectral density function plots of some line codes are analyzed. A free tool is used to implement the power spectral density functions in the time domain to validate their behavior in the frequency domain. This is done for the normalized case, where the frequency parameter (f) is equated to the bit rate parameter (R), and the value of the bit time parameter (Tb) is obtained as an inverse relation with the data rate.

Keywords: Data rate, Line codes, Free software, Power, Spectral density.

[1] BSc. in Electronic Engineering, Universidad Distrital Francisco José de Caldas, Colombia. Current position: Professor at Universidad Distrital Francisco José de Caldas, Colombia. E-mail: gacastangm@udistrital.edu.co ORCID: https://orcid.org/0000-0001-9788-5121

[2] Universidad Distrital Francisco José de Caldas, Faculty of Technology, Colombia. E-mail: rrodriguezg@udistrital.edu.co ORCID: https://orcid.org/0000-0002-2956-9650

[3] Universidad Distrital Francisco José de Caldas, Technology Faculty, Colombia. E-mail: cavanegas@udistrital.edu.co

Cite this article as: G. Castang-Montiel, R. Rodríguez-Guerrero, C. A. Vanegas, “Functions and codes for obtaining the normalized power spectral density with free software”, *Visión Electrónica*, vol. 16, no. 1, 2022. https://doi.org/10.14483/22484728.18886

Resumen

En este artículo se analizan los códigos implementados para la obtención de las gráficas de la función de la densidad espectral de potencia de algunos códigos de línea. Se utiliza una herramienta libre para implementar las funciones de la densidad espectral de potencia en el dominio del tiempo, para validar su comportamiento en el dominio de la frecuencia. Esto se realiza para el caso normalizado, donde el parámetro de la frecuencia (f) se iguala al parámetro tasa de bits (R), y se obtiene el valor del parámetro tiempo de bit (Tb) como una relación inversa con la tasa de datos.

Palabras clave: Tasa de datos, Códigos de línea, Software libre, Potencia, Densidad espectral.

1. Introduction

The use of free software tools for the analysis of signals, regarding their behavior from one domain to another is quite important, since it facilitates their understanding and the determination of their behavior in a particular domain, for example in the time or frequency domain, among others, as in the scale domain, when small waves or wavelets are used. The use of these tools for educational purposes is a good alternative, in which complex calculations and the graphing of signals with mathematical functions can be performed with great certainty and reliability in the results obtained and to be able to perform an adequate interpretation of a function or phenomenon to be analyzed.

2. Objectives

To show the coding process for plotting power spectral density functions, using the corresponding mathematical expressions, and performing the variation of the bit rate parameter, validating its incidence in the corresponding results.

3. Development methodology

In this case, the implementation or coding of the power spectral density functions, for some unipolar and bipolar, return-to-zero (RZ) and non-return-to-zero (NRZ) line codes, together with the two-phase or Manchester codes, is performed. The normalized case is analyzed, along with the variation of the bit rate parameter, and its implication on the resulting normalized power spectral density plots is validated [1].

The development interface used is Python, a free and very useful tool, which facilitates the development of this type of simulations and analysis. Its mathematical library is used to handle sine functions, the sinoc function, along with its squared values or raised to the power. It was also possible to implement the sigma function or unit impulse function and the product of these functions in the corresponding equations. The parameters taken for the graphing and analysis of the signals are the amplitude value (A), frequency (f), bit time (Tb), together with some constant values [2].

The functions implemented to obtain the power spectral density of the line codes used are listed below and are reflected in the respective lines of code.

3.1. Normalized power spectral density functions for the used line codes

This section contains the power spectral density functions and the normalized power spectral density functions for the line codes used. The normalized power spectral density function is obtained by dividing the power spectral density function by its maximum value, which is obtained by evaluating its autocorrelation function at zero [3].

3.1.1. Normalized power spectral density function for unipolar signals without return to zero

Its power spectral density is expressed as:

$$S(f) = \frac{A^2 * T_b}{4}\ senc^2(f *\ T_b)\left[1 + \frac{1}{T_b}\sigma(f)\right]$$

Where $\sigma(f)\ es\ la\ funcion\ delta\ o\ impulso\ unitario$; Likewise $A^2 = 2$.

Therefore, its normalized power spectral density is expressed as:

$$P_X(f) = \frac{T_b}{2}\ senc^2(f *\ T_b)\left[1 + \frac{1}{T_b}\sigma(f)\right]$$

3.1.2. Normalized power spectral density function for polar signals without zero return

Its power spectral density is expressed as:

$$S_X(f) = A^2\ T_b * senc^2(f\ T)$$

Likewise, $A^2 = 1$ and its normalized power spectral density is expressed as:

$$P_X(f) =\ T_b * senc^2(f\ T)$$

3.1.3. Normalized power spectral density function for unipolar signals with zero return

Its power spectral density is expressed as:

$$S(f) = \frac{A^2 * T_b}{16} senc^2\left(\frac{f * T_b}{2}\right)\left[1 + \frac{1}{T_b}\sum_{n=-\infty}^{\infty} \quad \sigma\left(f - \frac{n}{T_b}\right)\right]$$

Where $A^2 = 4$ where the normalized power spectral density is:

$$P_X(f) = \frac{T_b}{4} senc^2\left(\frac{f * T_b}{2}\right)\left[1 + \frac{1}{T_b}\sum_{n=-\infty}^{\infty} \quad \sigma\left(f - \frac{n}{T_b}\right)\right]$$

3.1.4. Normalized power spectral density function for bipolar signals with zero return

Its power spectral density is:

$$S(f) = \frac{A^2 * T_b}{4}\ senc^2\left(\frac{f * T_b}{2}\right) sen^2(\pi * f * T_b)$$

Where $A^2 = 4$ where the normalized power spectral density is:

$$P_X(f) = T_b\ senc^2\left(\frac{f * T_b}{2}\right) sen^2(\pi * f * T_b)$$

3.1.5. Normalized power spectral density function for Manchester encoded signals

Its power spectral density is:

$$S(f) = A^2 * T_b\ \ senc^2\left(\frac{f * T_b}{2}\right) sen^2\left(\pi * f * \frac{T_b}{2}\right)$$

Where $A^2 = 1$ where the normalized power spectral density is:

$$P_X(f) = T_b\ \ senc^2\left(\frac{f * T_b}{2}\right) sen^2\left(\pi * f * \frac{T_b}{2}\right)$$

In the implementation or development of each function in the code, the mathematical library of the software is used, together with the library for graphing, and the tool's function invocation method. The parameters that were defined for the amplitude, bit time and frequency parameters are used in each case.

The x-axis corresponds to the frequency, which extends up to a maximum value of twice the bit rate (2R), i.e., if a bit rate of one bit per second (1 bps) is used, as in the case of analysis 1A, the frequency values extend from the origin up to two Hertz (2 Hz). In the case of the y-axis, the normalized power spectral density is found, which will take a maximum value up to the value of the bit time, i.e., if the value of that parameter is one second (1 sec), the maximum normalized power value will be one watt (1 watt) [4].

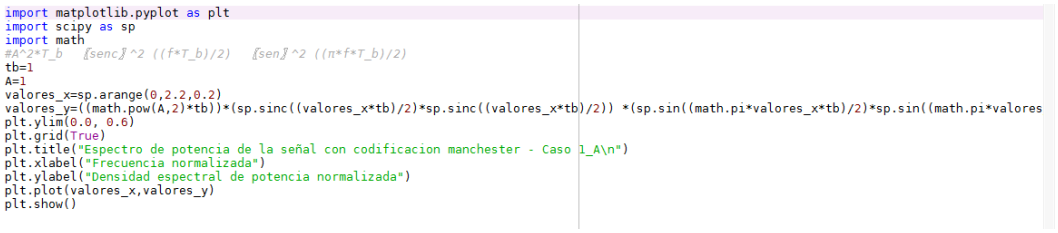
This change in the limits for the x-axis and y-axis in the graphs and in the code will be reflected when the bit rate parameter is varied, as in case 1-B, where a value of one thousand bits per second (1000 bps) is used, which affects the values reached in the resulting graphs.

4. Cases

4.1. Case 1-A

F=R; Tb=1/R. R= 1bps, f= 1Hz, Tb= 1sec.

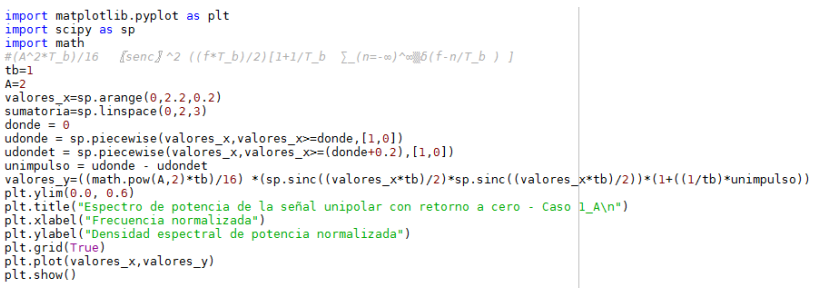
Figure 1. Code lines to obtain the power spectrum of the signal with Manchester coding.



```
import matplotlib.pyplot as plt
import scipy as sp
import math
#A^2*T_b   〖senc〗^2 ((f*T_b)/2)   〖sen〗^2 ((n*f*T_b)/2)
tb=1
A=1
valores_x=sp.arange(0,2.2,0.2)
valores_y=((math.pow(A,2)*tb))*(sp.sinc((valores_x*tb)/2)*sp.sinc((valores_x*tb)/2)) *(sp.sin((math.pi*valores_x*tb)/2)*sp.sin((math.pi*valores
plt.ylim(0.0, 0.6)
plt.grid(True)
plt.title("Espectro de potencia de la señal con codificacion manchester - Caso 1_A\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.plot(valores_x,valores_y)
plt.show()
```

In this case the bit time parameter is one second, and the amplitude is one volt. The frequency values are defined in the variable values_x, in an array of values from zero to two point two, with increment of sample values from zero point two. In the variable values_y, the power spectral density function is described, also on the y-axis is defined an excursion limit that goes from zero-to-zero point six, then the function is drawn with respect to the values obtained and shown with the corresponding labels. For the definition of the power spectral density function, the squared sine squared function and the squared sine function are used, considering the values of frequency and bit time.

Figure 2. Lines of code to obtain the power spectrum of the unipolar signal with return to zero



```
import matplotlib.pyplot as plt
import scipy as sp
import math
#(A^2*T_b)/16   〖senc〗^2 ((f*T_b)/2)[1+1/T_b  ∑_(n=-∞)^∞▒δ(f-n/T_b ) ]
tb=1
A=2
valores_x=sp.arange(0,2.2,0.2)
sumatoria=sp.linspace(0,2,3)
donde = 0
udonde = sp.piecewise(valores_x,valores_x>=donde,[1,0])
udondet = sp.piecewise(valores_x,valores_x>=(donde+0.2),[1,0])
unimpulso = udonde - udondet
valores_y=((math.pow(A,2)*tb)/16) *(sp.sinc((valores_x*tb)/2)*sp.sinc((valores_x*tb)/2))*(1+((1/tb)*unimpulso))
plt.ylim(0.0, 0.6)
plt.title("Espectro de potencia de la señal unipolar con retorno a cero - Caso 1_A\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.grid(True)
plt.plot(valores_x,valores_y)
plt.show()
```

In this function the bit time parameter is taken to be one second, the amplitude is two volts. The frequency values are taken in the variable values_x, taking values between zero and two with zero point two increments.

In this expression a summation is found with respect to the values of n, with the purpose of evaluating the delta function, which is displaced on the x-axis, varying its magnitude; in this case the integer values are taken for the displacement of the function between zero and two. In the variable y_values, the power spectral density function is defined, and an excursion limit is defined on the y-axis between zero and zero point six, then the function is drawn, and the corresponding labels are defined for each axis. For this case we have the squared sinoc function and the delta function, with respect to the frequency and bit time values.

Figure 3. Lines of code to obtain the power spectrum of the unipolar signal without return to zero

```
import matplotlib.pyplot as plt
import scipy as sp
import math
tb=1
A=math.pow(math.sqrt(2),2)
valores_x=sp.arange(0,2.2,0.2)
donde = 0
udonde = sp.piecewise(valores_x,valores_x>=donde,[1,0])
udondet = sp.piecewise(valores_x,valores_x>=(donde+0.2),[1,0])
unimpulso = udonde - udondet
valores_y=((A*tb)/4)*(sp.sinc(valores_x*tb) *sp.sinc(valores_x*tb)) *(1+((1/tb)*unimpulso))
plt.ylim(0.0, 1.1)
plt.title("Espectro de potencia de la señal unipolar sin retorno a cero - Caso_1_A\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.grid(True)
plt.plot(valores_x,valores_y)
plt.show()
```

For this case, the logic used in the previous cases is maintained with respect to the use of the variables, the bit time and amplitude values are one second and the amplitude squared is two volts. The frequency values on the x-axis and the delta function, which is determined at the origin, are established. The power spectral density function is defined on the y-axis, and the excursion limits of the function are set, then the labels for each axis are defined and the corresponding function is drawn. For this case, the squared sinoc function is taken, taking as arguments the frequency and bit time parameters.

Figure 4. Code lines to obtain the power spectrum of the bipolar signal with return to zero.

```
import matplotlib.pyplot as plt
import scipy as sp
import math
#(A^2*T_b)/4   〖senc〗^2 ((f*T_b)/2)  〖sen〗^2 (π*f*T_b )

tb=1
A=2
valores_x=sp.arange(0,2.2,0.2)
valores_y=((math.pow(A,2)*tb)/4)*(sp.sinc((valores_x*tb)/2)*sp.sinc((valores_x*tb)/2)) *(sp.sin(math.pi*valores_x*tb)*sp.sin(math.pi*valores_x*
plt.ylim(0.0, 1.1)
plt.grid(True)
plt.title("Espectro de potencia de la señal bipolar con retorno a cero - Caso 1_A\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.plot(valores_x,valores_y)
plt.show()
```

The bit time parameter is one second, and the amplitude value is two volts, so its amplitude value squared is four volts.

In this case the squared sine function and the squared sine function are used, having as arguments the values of frequency and bit time. Likewise, the signal is plotted on the y-axis, with respect to the evaluated values of the power spectral density function on the x-axis.

Figure 5. Code lines to obtain the power spectrum of the polar signal without return to zero

```
import matplotlib.pyplot as plt
import scipy as sp
tb=1
A=1
valores_x=sp.arange(0,2.2,0.2)
valores_y=((A*tb))*(sp.sinc(valores_x*tb) *sp.sinc(valores_x*tb))
plt.ylim(0.0, 1.1)
plt.grid(True)
plt.title("Espectro de potencia de la señal polar sin retorno a cero- Caso 1_A\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.plot(valores_x,valores_y)
plt.show()
```

The bit time value is one second, and the amplitude value is one volt. The squared sinoc function is used with respect to the frequency and bit time values. In the variable values_x, the frequency values are defined and in values_y the power density function is set. The resulting graph is obtained with respect to these two variables [5].

4.2. Case 1-B

F=R, Tb= 1/R, R= 1000bps, f= 1000 Hz, Tb= 1msec= 0.001 sec

The following is the case 1-B, which is normalized, keeping the frequency equal to the bit rate, and it is intended to evaluate the effect of the variation of the bit rate, in the normalized power

spectral density function plot, when the bit rate is greater than one thousand times, compared to the value of the bit rate, used in case 1-A.

In this case, the same procedure is maintained in the coding with respect to the previous case, evidencing a variation in the bit time, which goes from one second to one millisecond, and the frequency range is extended, which is from zero to two thousand, with increments of two. The maximum excursion values in the power spectral density function would then be up to a maximum value of one milliwatt.

The code for each case is attached, and the corresponding power density functions are evaluated, taking into account the variation of the frequency and bit time parameters; the bit time parameter is related to the data rate in an inverse relationship, i.e., the bit time decreases when the data rate increases and vice versa.

Figure 6. Code lines to obtain the power spectrum of the signal with Manchester coding

```
import matplotlib.pyplot as plt
import scipy as sp
import math
#A^2*T_b  〖senc〗^2 ((f*T_b)/2)  〖sen〗^2 ((π*f*T_b)/2)
tb=0.001
A=1
valores_x=sp.arange(0,2000,2)
valores_y=((math.pow(A,2)*tb))*(sp.sinc((valores_x*tb)/2)*sp.sinc((valores_x*tb)/2)) *(sp.sin((math.pi*valores_x*tb)/2)*sp.sin((math.pi*valores
plt.ylim(0.0, 0.0006)
plt.grid(True)
plt.title("Espectro de potencia de la señal con codificacion manchester - Caso 1-B\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.plot(valores_x,valores_y)
plt.show()
```

Figure 7. Lines of code to obtain the power spectrum of the unipolar signal with return to zero

```
import matplotlib.pyplot as plt
import scipy as sp
import math
#(A^2*T_b)/16  〖senc〗^2 ((f*T_b)/2)[1+1/T_b  ∑_(n=-∞)^∞▒δ(f-n/T_b ) ]
tb=0.001
A=2
valores_x=sp.arange(0,2000,2)
sumatoria=sp.linspace(0,2,3)
donde = 0
udonde = sp.piecewise(valores_x,valores_x>=donde,[1,0])
udondet = sp.piecewise(valores_x,valores_x>=(donde+0.2),[1,0])
unimpulso = udonde - udondet
valores_y=((math.pow(A,2)*tb)/16) *(sp.sinc((valores_x*tb)/2)*sp.sinc((valores_x*tb)/2))*(1+((1/tb)*unimpulso))
plt.ylim(0.0, 0.0009)
plt.title("Espectro de potencia de la señal unipolar con retorno a cero- Caso 1_B\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.grid(True)
plt.plot(valores_x,valores_y)
plt.show()
```

The amplitude parameter remains constant with respect to the previous case and according to the values determined for the normalized power spectral density, as evidenced above. This is in accordance with the existing theory for the analysis of the power spectral density of line codes.

Figure 8. Lines of code to obtain the power spectrum of the unipolar signal without return to zero

```
import matplotlib.pyplot as plt
import scipy as sp
import math
tb=0.001
A=math.pow(math.sqrt(2),2)
valores_x=sp.arange(0,2000,2)
donde = 0
udonde = sp.piecewise(valores_x,valores_x>=donde,[1,0])
udondet = sp.piecewise(valores_x,valores_x>=(donde+0.2),[1,0])
unimpulso = udonde - udondet
valores_y=((A*tb)/4)*(sp.sinc(valores_x*tb) *sp.sinc(valores_x*tb)) *(1+((1/tb)*unimpulso))
plt.ylim(0.0, 0.0009)
plt.title("Espectro de potencia de la señal unipolar sin retorno a cero - Caso_1_B\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.grid(True)
plt.plot(valores_x,valores_y)
plt.show()
```

Figure 9. Lines of code to obtain the power spectrum of the bipolar signal with return to zero

```
import matplotlib.pyplot as plt
import scipy as sp
import math
#(A^2*T_b)/4   〖senc〗^2 ((f*T_b)/2)  〖sen〗^2 (π*f*T_b )

tb=0.001
A=2
valores_x=sp.arange(0,2000,2)
valores_y=((math.pow(A,2)*tb)/4)*(sp.sinc((valores_x*tb)/2)*sp.sinc((valores_x*tb)/2)) *(sp.sin(math.pi*valores_x*tb)*sp.sin(math.pi*valores_x*
plt.ylim(0.0, 0.0009)
plt.grid(True)
plt.title("Espectro de potencia de la señal bipolar con retorno a cero - Caso 1_B\n")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.plot(valores_x,valores_y)
plt.show()
```

Figure 10. Lines of code to obtain the spectrum of the polar signal without return to zero

```
import matplotlib.pyplot as plt
import scipy as sp
tb=0.001
A=1
valores_x=sp.arange(0,2000,2)
valores_y=((A*tb))*(sp.sinc(valores_x*tb) *sp.sinc(valores_x*tb))
plt.ylim(0.0, 0.0011)
plt.grid(True)
plt.title("Espectro de potencia de la señal polar sin retorno a cero - Caso_1_B")
plt.xlabel("Frecuencia normalizada")
plt.ylabel("Densidad espectral de potencia normalizada")
plt.plot(valores_x,valores_y)
plt.show()
```

5. Results

The mathematical calculations required to obtain the power spectral density can be performed using specific functions such as the sine-square function, the sine-square function and the dirac delta function.

These functions can be properly implemented, together with the frequency parameters and the corresponding normalized power spectral density function in each case, to obtain and analyze the corresponding results.

These analyses allow to corroborate the existing theory, and also to validate the effects of the direct current components, for the frequency value equal to zero, and the power spectral density waveforms in which the effects of these components can be mitigated, or their effects are not significant.

The effect of the harmonic components of the power spectral density function can also be found in each case.

6. Conclusions

The use of free tools allows the analysis of functions and signals in an appropriate way, using the corresponding mathematical editors.

The use and diffusion of this type of tools in educational and academic environments allows a better understanding and compression of signals and functions, by means of analysis and experimentation, through simulation.

Acknowledgments

We thank God and the Universidad Distrital Francisco José de Caldas, for giving us the opportunity to grow as people, in the interaction with our students, colleagues and academic community in general, in a process of continuous teaching and learning.

References

[1] S. Guerrero, C. Vanegas, G. Castang, “Python a su alcance”, Bogotá: UD editorial, 2020.

[2] L. W. Couch, “Power spectrum of binary line codes”, Fifth Edition, Pearson Education, pages 159-166.

[3] J. C. Martínez-Quintero, E. P. Estupiñán-Cuesta, V. D. Rodríguez-Ortega, “Raspberry PI 3 RF signal generation system”, Visión electrónica, vol. 13, no. 2, pp. 294–299, 2019. https://doi.org/10.14483/22484728.15160

[4] Cacheme, “Python course for engineers”. [online]. Available: https://cacheme.org//curso-online-python-cientifico-ingenieros/

[5] Matpic, “Python examples”. [online]. Available: http://matpic1984.blogspot.com/2015/08/serie-de-fourier-de-una-onda-cuadrada.html