Hands-on 4 (parte 02): Implementação via Python para plotagem de BER vs SNR

Vicente Sousa
GppCom/DCO/UFRN

Natal, 05/10/2016

Universidade Federal do Rio Grande do Norte (UFRN)

Objetivos

- Explicar a função do arquivo *berawgn.py (gnuradio/gr-digital/examples).*
- Comparar o script *berawgn.py* e a simulação realizada no hands-on 4 do minicurso (via GRC).

Descrição

- O script que será apresentado aqui tem como finalidade exibir um gráfico contendo o resultado de uma simulação de um número N de amostras de entrada em um sistema de transmissão BPSK, para valores de SNR(dB) pré-definidos.

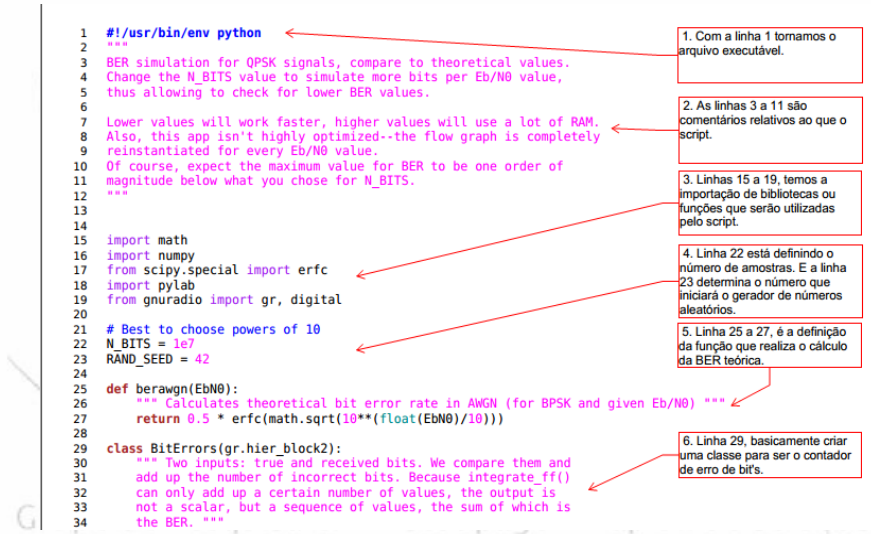
© Prof. Dr. Vicente Angelo de Sousa Junior @ GppCom - UFRN
vicente.sousa@ct.ufrn.br

Descrição

- O script tem o nome: *berawgn.py*.
- Clique com o botão direito do mouse sobre o arquivo e siga a sequência abaixo:
  - Abrir ⟶ Display;
- Com isso o arquivo será aberto no gedit do linux.
- O código será comentado a seguir.

© Prof. Dr. Vicente Angelo de Sousa Junior @ GppCom - UFRN
vicente.sousa@ct.ufrn.br

Código linha a linha



```
1  #!/usr/bin/env python
2  """
3  BER simulation for QPSK signals, compare to theoretical values.
4  Change the N_BITS value to simulate more bits per Eb/N0 value,
5  thus allowing to check for lower BER values.
6
7  Lower values will work faster, higher values will use a lot of RAM.
8  Also, this app isn't highly optimized--the flow graph is completely
9  reinstantiated for every Eb/N0 value.
10 Of course, expect the maximum value for BER to be one order of
11 magnitude below what you chose for N_BITS.
12 """
13
14
15 import math
16 import numpy
17 from scipy.special import erfc
18 import pylab
19 from gnuradio import gr, digital
20
21 # Best to choose powers of 10
22 N_BITS = 1e7
23 RAND_SEED = 42
24
25 def berawgn(EbN0):
26     """ Calculates theoretical bit error rate in AWGN (for BPSK and given Eb/N0) """
27     return 0.5 * erfc(math.sqrt(10**(float(EbN0)/10)))
28
29 class BitErrors(gr.hier_block2):
30     """ Two inputs: true and received bits. We compare them and
31     add up the number of incorrect bits. Because integrate_ff()
32     can only add up a certain number of values, the output is
33     not a scalar, but a sequence of values, the sum of which is
34     the BER. """
```

1. Com a linha 1 tornamos o arquivo executável.

2. As linhas 3 a 11 são comentários relativos ao que o script.

3. Linhas 15 a 19, temos a importação de bibliotecas ou funções que serão utilizadas pelo script.

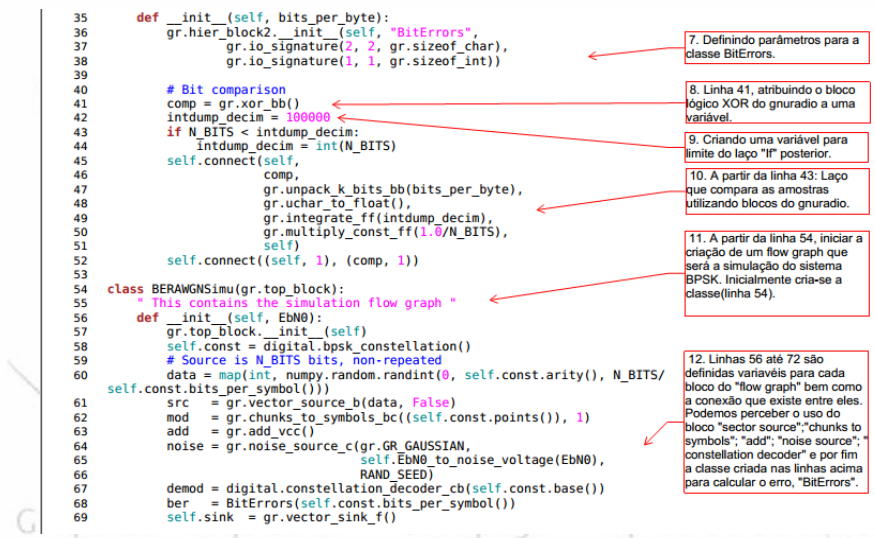
4. Linha 22 está definindo o número de amostras. E a linha 23 determina o número que iniciará o gerador de números aleatórios.

5. Linha 25 a 27, é a definição da função que realiza o cálculo da BER teórica.

6. Linha 29, basicamente criar uma classe para ser o contador de erro de bit's.

© Prof. Dr. Vicente Angelo de Sousa Junior @ GppCom - UFRN
vicente.sousa@ct.ufrn.br

Código linha a linha



```
35     def __init__(self, bits_per_byte):
36         gr.hier_block2.__init__(self, "BitErrors",
37                 gr.io_signature(2, 2, gr.sizeof_char),
38                 gr.io_signature(1, 1, gr.sizeof_int))
39
40         # Bit comparison
41         comp = gr.xor_bb()
42         intdump_decim = 100000
43         if N_BITS < intdump_decim:
44             intdump_decim = int(N_BITS)
45         self.connect(self,
46                      comp,
47                      gr.unpack_k_bits_bb(bits_per_byte),
48                      gr.uchar_to_float(),
49                      gr.integrate_ff(intdump_decim),
50                      gr.multiply_const_ff(1.0/N_BITS),
51                      self)
52         self.connect((self, 1), (comp, 1))
53
54 class BERAWGNSimu(gr.top_block):
55     " This contains the simulation flow graph "
56     def __init__(self, EbN0):
57         gr.top_block.__init__(self)
58         self.const = digital.bpsk_constellation()
59         # Source is N_BITS bits, non-repeated
60         data = map(int, numpy.random.randint(0, self.const.arity(), N_BITS/
    self.const.bits_per_symbol()))
61         src   = gr.vector_source_b(data, False)
62         mod   = gr.chunks_to_symbols_bc((self.const.points()), 1)
63         add   = gr.add_vcc()
64         noise = gr.noise_source_c(gr.GR_GAUSSIAN,
65                                   self.EbN0_to_noise_voltage(EbN0),
66                                   RAND_SEED)
67         demod = digital.constellation_decoder_cb(self.const.base())
68         ber   = BitErrors(self.const.bits_per_symbol())
69         self.sink  = gr.vector_sink_f()
```

7. Definindo parâmetros para a classe BitErrors.

8. Linha 41, atribuindo o bloco lógico XOR do gnuradio a uma variável.

9. Criando uma variável para limite do laço "If" posterior.

10. A partir da linha 43: Laço que compara as amostras utilizando blocos do gnuradio.

11. A partir da linha 54, iniciar a criação de um flow graph que será a simulação do sistema BPSK. Inicialmente cria-se a classe(linha 54).

12. Linhas 56 até 72 são definidas variavéis para cada bloco do "flow graph" bem como a conexão que existe entre eles. Podemos perceber o uso do bloco "sector source";"chunks to symbols"; "add"; "noise source"; " constellation decoder" e por fim a classe criada nas linhas acima para calcular o erro, "BitErrors".

© Prof. Dr. Vicente Angelo de Sousa Junior @ GppCom - UFRN
vicente.sousa@ct.ufrn.br

Código linha a linha

```
70          self.connect(src, mod, add, demod, ber, self.sink)
71          self.connect(noise, (add, 1))
72          self.connect(src, (ber, 1))
73
74      def EbN0_to_noise_voltage(self, EbN0):
75          """ Converts Eb/N0 to a single-sided noise voltage (assuming unit symbol power) """
76          return 1.0 / math.sqrt(2.0 * self.const.bits_per_symbol() * 10**(float(EbN0)/10))
77
78
79  def simulate_ber(EbN0):
80      """ All the work's done here: create flow graph, run, read out BER """
81      print "Eb/N0 = %d dB" % EbN0
82      fg = BERAWGNSimu(EbN0)
83      fg.run()
84      return numpy.sum(fg.sink.data())
85
86  if __name__ == "__main__":
87      EbN0_min = -3
88      EbN0_max = 10
89      EbN0_range = range(EbN0_min, EbN0_max+1)
90      ber_theory = [berawgn(x)      for x in EbN0_range]
91      print "Simulating..."
92      ber_simu   = [simulate_ber(x) for x in EbN0_range]
93
94      f = pylab.figure()
95      s = f.add_subplot(1,1,1)
96      s.semilogy(EbN0_range, ber_theory, 'g-.', label="Theoretical")
97      s.semilogy(EbN0_range, ber_simu, 'b-o', label="Simulated")
98      s.set_title('BER Simulation')
99      s.set_xlabel('Eb/N0 (dB)')
100     s.set_ylabel('BER')
101     s.legend()
102     s.grid()
103     pylab.show()
```

13. Criando função para converter SNR(dB) em SNR liner considerando amplitude do sinal unitária.

14. Gerando função que chama a classe BERAWGNSimu crianda anteriormente (o flow graph), roda-a e lê a saída da BER.

15. A partir da linha 86: definição do intervalo de SNR (dB) a ser analisado.

16. Comandos de plotagem dos resultados da simulação.

© Prof. Dr. Vicente Angelo de Sousa Junior @ GppCom - UFRN
vicente.sousa@ct.ufrn.br

Comparação com o GRC

- Paralelo entra a simulação realizada pelo hands-on 4 e o script mostrado nessa apresentação.

Ação	Hands-on 4 (Blocos)	Script: *berawgn.py* (Linhas)
Geração de dados	random source	60 e 61
Esquema da modulação	modulação: chunks to symbols+add+noise source; demodulação: complex to real+binary slicer	62 à 67
Cálculo de conversão de SNR	O cálculo de conversão é feito no cálculo da amplitude do ruído no bloco: noise source	25 à 27
Cálculo da amplitude do ruído	noise source	74 à 76
Cálculo da BER	throttle+error rate+ number sink	29 à 52

© Prof. Dr. Vicente Angelo de Sousa Junior @ GppCom - UFRN
vicente.sousa@ct.ufrn.br

Execução do script

- Para executa o script:
  - Abra um terminal e navegue até a pasta onde se encontra o arquivo, ou até a área de trabalho do linux de sua máquina virtual. Digite os seguintes comandos:

    # python *berawgn.py*
  - A simulação do script iniciará. Observe a figura a seguir.

© Prof. Dr. Vicente Angelo de Sousa Junior @ GppCom - UFRN
vicente.sousa@ct.ufrn.br

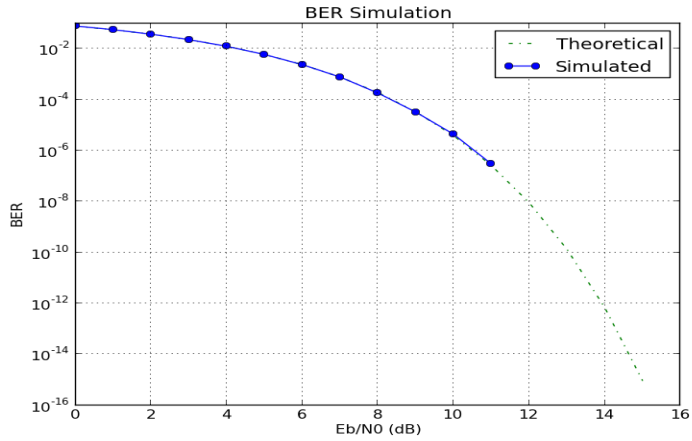
Execução do script

```
minicurso@ubuntu: ~/Desktop
minicurso@ubuntu:~$ cd Desktop/
minicurso@ubuntu:~/Desktop$ python berawgn_BPSK.py
Simulating...
Eb/N0 = -5 dB
Using Volk machine: sse4_a_32_orc
Eb/N0 = -4 dB
Eb/N0 = -3 dB
Eb/N0 = -2 dB
Eb/N0 = -1 dB
Eb/N0 = 0 dB
Eb/N0 = 1 dB
Eb/N0 = 2 dB
Eb/N0 = 3 dB
Eb/N0 = 4 dB
Eb/N0 = 5 dB
Eb/N0 = 6 dB
Eb/N0 = 7 dB
Eb/N0 = 8 dB
Eb/N0 = 9 dB
Eb/N0 = 10 dB
Eb/N0 = 11 dB
Eb/N0 = 12 dB
minicurso@ubuntu:~/Desktop$
```


© Prof. Dr. Vicente Angelo de Sousa Junior @ GppCom - UFRN
vicente.sousa@ct.ufrn.br

Execução do script

- Terminada a simulação o resultado será similar a figura abaixo.



Comentários finais

- Pode-se destacar a relação forte entre o GRC e o python;
- GRC gerar código legível em python pode facilitar a extensão de funcionalidade do código gerado com o GRC
  - Scripts em python pode ser melhores para realizar campanhas de simulações.

Sobre o GppCom

- **A meta do GppCom é criar na UFRN um ambiente de P&D&I através de prototipagem rápida baseada em simulação via software e hardware nas áreas de sistemas de comunicação e processamento digital de sinais e imagens. O Grupo é formado pelos professores: Vicente Angelo de Sousa Junior (coordenador), Luiz Gonzaga de Queiroz Silveira Junior (vice-coordenador), Luiz Felipe de Queiroz Silveira, Marcio Eduardo da Costa Rodrigues, Adaildo Gomes D'Assunção (pesquisador associado), Cláudio Rodrigues Muniz da Silva (pesquisador associado), Cristhianne de Fátima Linhares de Vasconcelos (pesquisador associado). O GppCom está de portas abertas para novas parcerias, conheça o portfólio do grupo.**

- **Contato:** vicente.gppcom@gmail.com

Grupo de Pesquisa em Prototipagem Rápida de Soluções para Comunicação.