Polar Codes in Python: Quick Reference

Brendon McBain

October 2019

Contents

1 Introduction

A quick reference for the *Polar Codes in Python* library that describes the classes and functions available to the user. An overview of the classes and their dependencies is shown in Figure 1. Section 2 of this document describes each class, including their functions and public variables. Section 3 is dedicated to a few simple examples to help new users intialise polar codes, specify a polar code construction, encode & decode, and simulate their codes. A more detailed version of the function implementations can be found in *Polar Codes in Python: Main Reference*.

It provides:

- non-systemic encoder and Successive Cancellation Decoder (SCD) for polar codes
- mothercode construction of polar codes using Bhattacharyya Bounds or Gaussian Approximation
- support for puncturing and shortening
- Bit-Reversal Shortening (BRS), Wang-Liu Shortening (WLS), and Bioglio-Gabry-Land (BGL) shortening constructions
- an AWGN channel with BPSK modulation

2 Getting Started

1. Download the main library package from https://github.com/mcba1n/polar-codes, and unzip it to "*root*".
2. Install *matplotlib* from https://matplotlib.org/users/installing.html
3. Install *numpy* from https://docs.scipy.org/doc/numpy/user/install.html.
4. Run *test.py* using a *Python*3 compiler. If the program runs successfully, the library is ready to use. Make sure the compiler has writing access to directory "*root/data*", where simulation data will be saved by default.
5. Run *main.py* to start the GUI.

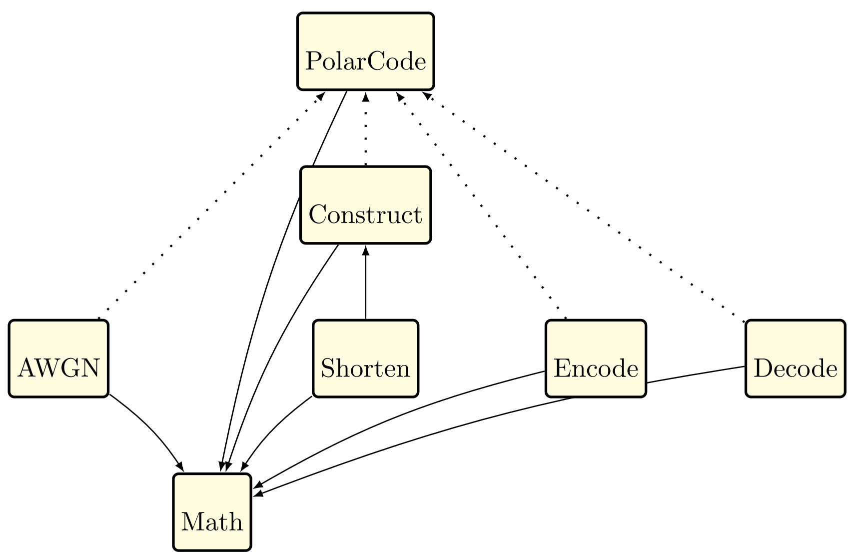

Figure 1: An overview of the classes in the library. The solid lines point to super-classes of the source (inheritance), and dashed lines point to classes that must be instantiated and given to the source class.

3 Class Definitions

3.1 PolarCode

An object that encapsulates all of the parameters required to define a polar code [1]. This object must be given to the following classes: AWGN, Construct, Decode, Encode, GUI, and Shorten.

```
# Example: Create a (256,100) mothercode
PolarCode(256,100)
```

PolarCode	
$PolarCode(N, K)$	
Public Variable	**Description**
N	The mothercode block length. N and M are the same if there is no puncturing.
M	The block length.
n	The number of bits per channel index, $log_2(N)$.
K	The code dimension.
$frozen$	The frozen indices.
$frozen\_lookup$	A look-up table for the frozen indices. Each "0" corresponds to a frozen bit index, and each "1" corresponds to an information bit index.
x	The uncoded message with parity bits.
$construction\_type$	The mothercode construction type, with options specified by $construction\_algos$. Set to "bb" by default.
$construction\_algos$	The supported mothercode construction algorithms: Bhattacharyya Bounds is referred to by "bb", and Gaussian Approximation is referred to by "ga".
$message\_received$	The decoded message from class Decode will be stored here.
$punct\_type$	The type of puncturing. Set "punct" for log-likelihoods of punctured bits to be set to zero, and "shorten" for log-likelihoods of shortened bits to be set to infinity.
$punct\_flag$	A flag that is set when N and M are not equal, hence puncturing is required.
$punct\_set$	The coded punctured bits.
$punct\_set\_lookup$	A look-up table for the frozen $punct\_set$. Each "0" corresponds to a punctured bit index, and each "1" corresponds to an information bit index.
s	The number of bits punctured in the mothercode, $N - M$.
$punct\_algorithm$	The puncturing algorithm, with options as specified by shortening_algos (this library does not include any standard puncturing algorithms).
$shortening\_algos$	The supported shortening algorithms: BRS, WLS, BGL, and Permutation.
$recip\_flag$	A flag set to True if the coded punctured bits are equal to the uncoded punctured bits, hence they are the so-called "reciprocal" puncturing patterns. Set to False by default.

Public Function	Description
$set\_message(m)$	Sets $PolarCode.x$ using the message vector input and $PolarCode.frozen$.
$get\_normalised\_SNR$ $(design\_SNR)$	Scales E_b/N_o by M/K, so that different rate codes can be compared. The input and output are in decibels.
$simulate$ $(sim\_filename, Eb\_No\_vec$ $[, design\_SNR = 5, max\_iter = 100000,$ $manual\_const\_flag = False, min\_iterations = 1000,$ $min\_errors = 30, sim\_seed = 1729])$	Simulate the polar code with a construction using a vector of E_b/N_o values (in decibels). The inputs also include the the max. number of iterations per Eb_No_vec value, min. number of iterations, min. number of frame errors, and the seed used to generate the pseudo-random messages (default seed is 1729, or "twister" in MATLAB).
$plot(sim\_filenames, dir)$	Plot all the files in the list $sim\_filenames$ from the directoy dir.

3.2 Construct

Construct performs the mothercode construction. It uses the algorithm specified by $PolarCode.construction\_type$. If the puncturing flag $PolarCode.punct\_flag$ is True the input PolarCode instance will go to class Shorten. Mothercode constructions supported: Bhattacharyya Bounds [1, 2], and Gaussian Approximation [3].

```
# Example: Construct the polar code with parameters specified in myPC and E_b/N_o = 5dB
Construct(myPC, 5)
```

Construct	
$Construct(myPC, design\_SNR[, manual = False])$	

Public Function	Description
$general\_pcc(z0)$	Polar code construction using Bhattacharyya Bounds. z0 is a vector of the initial Bhattacharyya parameters for N bit-channels in the log-domain. For non-punctured channels, the parameters are $-E_b/N_o$. It is adapted for the infinite likelihoods used in shortening.
$general\_ga(z0)$	Polar code construction using Density Evolution/Gaussian Approximation. z0 is a vector of the initial mean likelihood densities for N bit-channels in the log-domain. For non-punctured channels, the parameters are $4E_b/N_o$.
$perfect\_pcc(p)$	A boolean expression approach to puncturing pattern construction [6]. Input p is a look-up table for the coded puncturing bits, and the look-up table for the uncoded punctured bits is returned. For shortening, take the complement of p and the output vector.

3.3 Shorten

A class dedicated to shortening. This is only directly useful for programmers who are doing manual construction, since Construct will instantiate this class if *punct_type* is "shorten". This means that the likelihoods for each coded shortened bit are set to infinity at the channel output given by class AWGN. Shortening techniques supported: WLS [4], BRS [5], BGL [7], and permutations of WLS.

Shorten	
$Shorten(myPC)$	
Public Function	**Description**
$last\_pattern()$	Returns the most common shortening pattern of the Wang-Liu Algorithm [4], $S_{WLS} = \{N-M, N-M+1, ..., N-1\}$. These are the last $PolarCode.s$ indices.
$brs\_pattern()$	Returns the bit-reversal of the last $PolarCode.s$ indices [5], $S_{BRS} = \{b(N-M), b(N-M+1), ..., b(N-1)\}$
$bgl\_pattern()$	Returns the shortening pattern from the BGL shortening algorithm [7].
$frozen\_from\_pattern$ $(punct\_set)$	Assumes a reciprocal shortening pattern and forces the frozen bits to include the shortening set. If $\mathcal{F}_0$ is the $M-K$ least reliable bit-channels, and $\mathcal{S}$ is the shortening set, then it sets $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{S}$. It uses the reliabilities from $PolarCode.reliabilities$, and returns the frozen set.
$perm()$	Returns a permutation of the bit indices of $\mathcal{S}_{WLS}$, as specified by $PolarCode.perm$. The permutation should be a list of non-repeated integers between 0 and $n-1$. Use this to generate new reciprocal patterns.

3.4 Encode

A polar encoder class. Currently only non-systematic encoding is supported. Ensure that $PolarCode.x$ is set by using $PolarCode.set\_message(m)$, and that the mothercode construction is described by $PolarCode.frozen$.

```
# Example: Assuming construction by class Construct, encode a random message.
my_message = np.random.randint(2, size=myPC.K)
myPC.set_message(my_message)
Encode(myPC)
```

Encode	
$Encode(myPC)$	
Public Function	**Description**

3.5 Decode

A polar decoder class. Currently only Successive Cancellation Decoder (SCD) is supported. Ensure that the output likelihoods of the channel are stored in *PolarCode.likelihoods* using the class AWGN.

```
# Example: Assuming myPC.likelihoods was set by class AWGN,
#          decode the message sent through the channel using SCD
Decode(myPC, 'scd')
```

Decode	
$Decode(myPC[, decoder\_name = "scd"])$	
Public Function	**Description**

3.6 AWGN

Simulates an AWGN channel by adding Gaussian noise with double-sided noise power N_o to a signal of energy E_s. This class updates *PolarCode.likelihoods* with the log-likelihood ratios for each bit of the codeword *PolarCode.u*. For puncturing, the likelihoods for the punctured bits in *PolarCode.punct_set_lookup* will be set to zero. For shortening, the likelihoods for the shortened bits in *PolarCode.punct_set_lookup* will be set to infinity. Currently only BPSK modulation is supported.

```
# Example: Simulate a channel for the message in myPC.u with E_b/N_o=5 dB
AWGN(myPC, 5)
```

AWGN	
$AWGN(myPC, Eb\_No[, plot\_noise = False])$	
Public Function	**Description**

3.7 Math

This class is intended for programmers who wish to implement their own functions for the library and require some common numerical methods. Many are also used by existing functions.

Math
$Math()$

Public Function	**Description**
$bit\_reversed(x, n)$	Returns the bit-reversal of an index x.
$logdomain\_sum(x, y)$	Returns the result of $x + y$ in the log-domain. The arguments x and y must be in the log-domain.
$logdomain\_diff(x, y)$	Returns the result of $x - y$ in the log-domain. The arguments x and y must be in the log-domain.
$bit\_perm(x, p, n)$	Returns the permutation of an index x, where p is the permutation and n is the number of bits in the index.
$hamming\_wt(x, n)$	Returns the bit-wise hamming weight of an index x.
$sort\_by\_wt(x, n)$	Returns a sorted vector by index hamming weights using $hamming\_wt()$ of a list x.
$inverse\_set(F, N)$	Returns $\{0, 1, ..., N - 1\} \setminus F$.
$arikan\_gen(n)$	Returns the n-th kronecker product of $F = [[1, 1], [0, 1]]$, commonly referred to as Arikan's kernel. Hence, an encoded codeword would be $u_N = F_N x_N$.

4 Examples

4.1 Mothercode

An example of encoding and decoding over an AWGN channel for a (64,32) mothercode, using Bhattacharyya Bounds and SCD.

```
import numpy as np
from classes.PolarCode import PolarCode
from classes.Construct import Construct
from classes.Encode import Encode
from classes.Decode import Decode
from classes.AWGN import AWGN

# initialise polar code
myPC = PolarCode(64, 32)

# mothercode construction
design_SNR  = 5.0
Construct(myPC, design_SNR)
print(myPC, "\n\n")

# set message
my_message = np.random.randint(2, size=myPC.K)
myPC.set_message(my_message)
print("The message is:", my_message)

# encode message
Encode(myPC)
print("The coded message is:", myPC.x)

# transmit the codeword
AWGN(myPC, design_SNR)
print("The log-likelihoods are:", myPC.likelihoods)

# decode the received codeword
Decode(myPC)
print("The decoded message is:", myPC.message_received)
```

4.2 Shortened Code

An example of encoding and decoding a (200,100) shortened code, using the Gaussian Approximation mothercode construction, Bit-Reversal Shortening (BRS), and SCD.

```
import numpy as np
from classes.PolarCode import PolarCode
from classes.Construct import Construct
from classes.Encode import Encode
```

```
from classes.Decode import Decode
from classes.AWGN import AWGN

# initialise polar code
myPC = PolarCode(200, 100)
myPC.construction_type = 'ga'
myPC.punct_type = 'shorten'
myPC.punct_algorithm = 'brs'

# construction
design_SNR  = 5.0
Construct(myPC, design_SNR)
print(myPC, "\n\n")

# set message
my_message = np.random.randint(2, size=myPC.K)
myPC.set_message(my_message)
print("The message is:", my_message)

# encode message
Encode(myPC)
print("The coded message is:", myPC.x)

# transmit the codeword
AWGN(myPC, design_SNR)
print("The log-likelihoods are:", myPC.likelihoods)

# decode the received codeword
Decode(myPC)
print("The decoded message is:", myPC.message_received)
```

4.3 Simulation & Plotting

A script to simulate a defined polar code, save the data to a *JSON* file in directory "/data", and then display the result in a *matplotlib* figure. By default, the simulation will have an early stopping condition of 30 minimum frame errors and 1,000 minimum errors for each E_b/N_o, as well as a maximum of 100,000 simulations.

```
from classes.PolarCode import PolarCode
import numpy as np

# initialise polar code
myPC = PolarCode(64, 32)

# simulate polar code (default settings)
myPC.simulate('data/pc_sim', np.arange(1,5))
```

```
# plot the frame error rate
myPC.plot(['pc_sim'], 'data/')
```

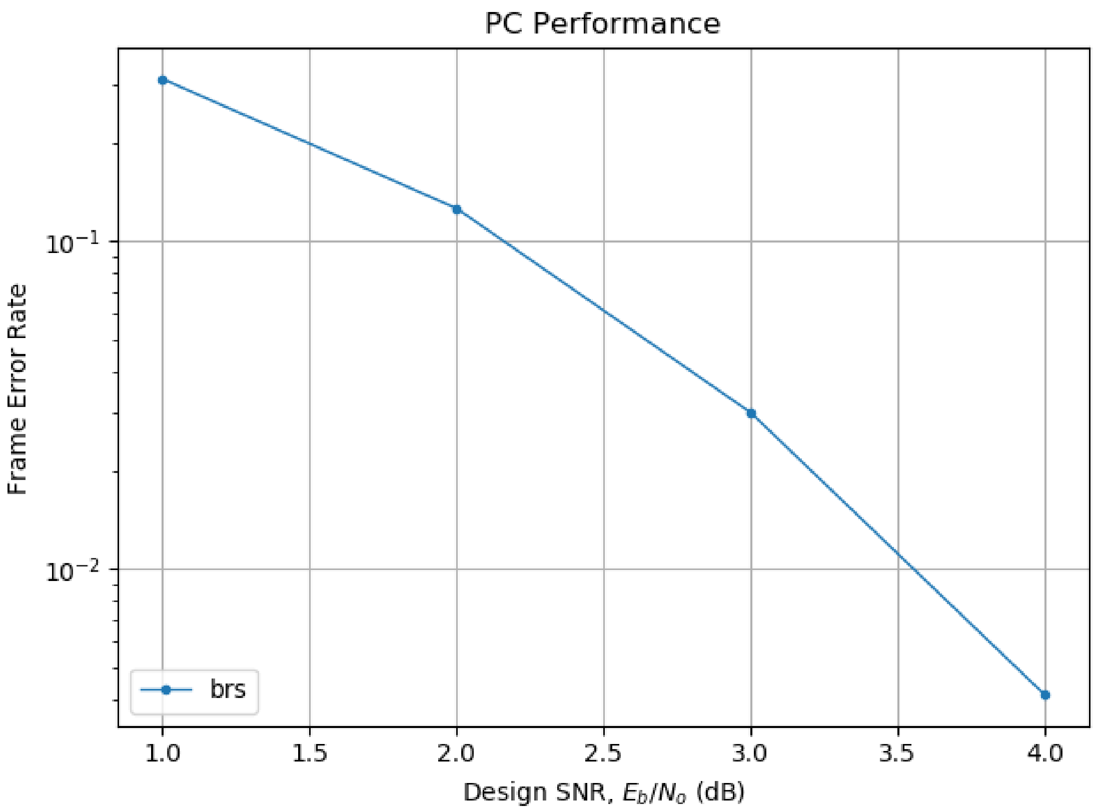

Figure 2: A screenshot of the output for the example in Section 4.3.

5 Graphical User Interface

The polar codes library has an accompanying Graphic User Interface (GUI) for users who wish to quickly simulate a code, without any knowledge of Python. Previously, this option has not been available to the community of polar codes. Figure 3 shows an example simulation and plot of a (64,32) mothercode.

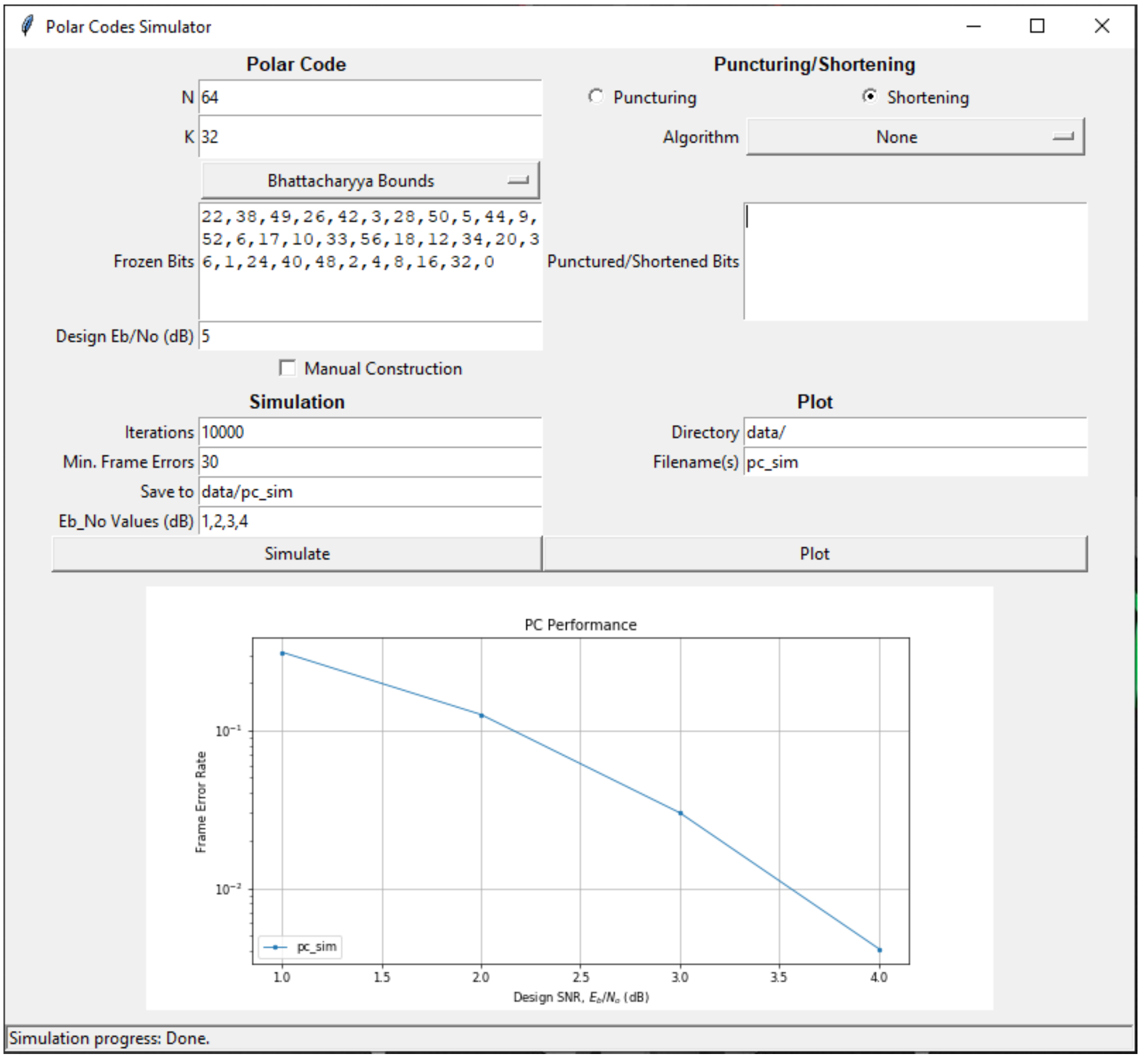

Figure 3: A screenshot of the GUI.

5.1 Input Fields

The only fields that are absolutely required to be filled in by the user are N, K, *Construction*, and *Design* E_b/N_o, in the case of mothercodes. The default values in the rest of the fields should be sufficient to simulate the code. For a punctured code, in addition to the previous fields, the user must select a shortening algorithm from *Algorithm*, and select the "Puncturing" or "Shortening" (Note: the only algorithms available are for shortening, so selecting "Puncturing" will revert back to the mothercode construction). However, if *Manual Construction* is ticked, the text fields *Frozen Bits* and *Punctured/Shortening Bits* must be specified by user. Otherwise, these would normally be set by the simulator upon completion of the simulation using the construction algorithms selected on the interface.

5.2 Saving & Plotting Data

The GUI also lets the user save their simulations to a directory specified by *Save to*. The user can save multiple datasets to the same directory, and then plot them all by specifying the root directory in *Directory* and a list of filenames in *Filename(s)*. Each dataset will be labelled on a legend using their filenames.

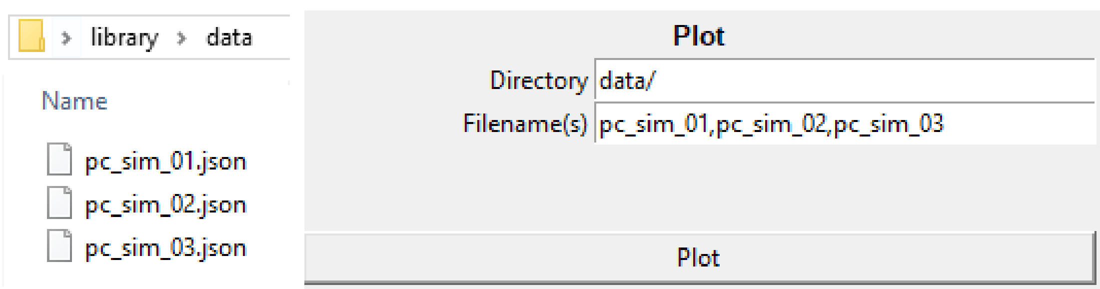

Figure 4: The directory setup for plotting multiple datasets on the same axes.

References

[1] Erdal Arikan. *Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels*. IEEE Transactions on Information Theory, Vol. 55, No. 7, July 2009.

[2] Harish Vangala, Emanuele Viterbo, and Yi Hong. *Permuted Successive Cancellation Decoder*. ISITA2014, Melbourne, Australia, October 26-29, 2014.

[3] Harish Vangala, Emanuele Viterbo, and Yi Hong. *A Comparative Study of Polar Code Constructions for the AWGN Channel*. arXiv:1501.02473 [cs.IT], 2015.

[4] Runxin Wang and Rongke Liu. *A novel puncturing scheme for polar codes*. IEEE Communications Letters, Vol. 18, No. 12, December 2014.

[5] Prakash Chaki and Norifumi Kamiya. *On the properties of bit-reversal shortening in polar codes*. ISITA2018, Singapore, October 28-31, 2018.

[6] Song-Nam Hong and Min-Oh Jeong. *On the analysis of puncturing for finite-length polar codes: Boolean function approach*. IEEE Transactions on Communications, Vol. 66, No. 11, November 2018.

[7] Valerio Bioglio, Frederic Gabry, and Ingmar Land. *Low-Complexity Puncturing ans Shortening of Polar Codes*. arXiv:1701.06458v1 [cs.IT] 23 Jan 2017.