Getting started with LDPC codes on Windows

A whitepaper by Jeremias Pohle - Group 5, with footnote by Max Iyengar - Group 3

This whitepaper aims to get Windows users started on compiling and running Jossy Sayir's LDPC code package, downloadable from here.

Installation of MSYS2:

MSYS2 is a widely used C development package. It can be installed through an installer, downloaded from https://www.msys2.org. Following this a few commands have to be run in the MSYS2 terminal to install the required compiler:

```
pacman -Suy
```

This will close the terminal window. Reopen it.

```
pacman -Suy
pacman -S mingw-w64-x86_64-gcc
```

(This installs the compiler required to turn the C sourcecode into .dll files that can be run from Python)

Close the terminal and open the newly accessible MSYS2 MINGW64 terminal (simply press the Win key and start typing MSYS2 MINGW64).

Verify that the compiler has properly installed by running `gcc --version`
This command should return without errors if the compiler is installed properly. Note that running this command in the terminal or the root MSYS2 shell will return an error. Hence, ensure that you are using the MINGW64 shell!

Compiling the C source code:

Unfortunately, the C code compiles into a dynamic link library. This has the advantage that it can contain a vast amount of different data, but it also makes the compiled code dependent on local libraries. These are contained in the MSYS home directory. This also means that the compiler can only compile within the MSYS home directory, as well as run files contained in the home directory.
To compile the source code, first copy the lpdc_jossy folder extracted from the zip file retrieved above into the MSYS home directory. This can be found under

```
C:/msys64/home/user_name
```

Note that this directory is only accurate if no changes were made to the installation directory. Choose the correct directory if you did make changes to the installation location.
Once the folder is in place, open the MSYS2 MINGW64 shell and navigate to the copied folder using `cd` commands appropriately. (TIP: use `dir` to return a list of all the folders and files in the current directory!).
Following this, the compilation commands are similar to the ones Jossy provides in the README of the package. However, the code has to be compiled to a .dll file instead of a .so

file (.dll is the Windows equivalent of .so (shared object) in Linux and MacOS systems). Thus, run:

```
mkdir bin
```

(the package does not come with an empty folder called bin, it has to be created by the user using the command above!)

```
gcc -lm -shared -fPIC -o bin/c_ldpc.dll src/c_ldpc.c
gcc -o bin/results2csv src/results2csv.c
```

These two lines compile the relevant code into a .dll and executable respectively.

IT IS RECOMMENDED TO NOW COPY THIS FOLDER INTO YOUR PROJECT DIRECTORY. THIS SHORTENS THE PATH FOR IMPORTING THE PACKAGE.

Modifying the Python code:

Unfortunately, there seems to have been changes in the way some Python libraries use data-types since this package was published. Line 394 of the ldpc.py file sorts a list. This returns in a data type unsuitable for handling in later functions. Hence, insert this line below it:

```
intrlv = intrlv.astype(np.int32)
```

Also, some of the steps executed above demand the Python code to be changed slightly. The directory and file type in lines 467, 493 and 498 have to be changed accordingly. Ensure that you use the .dll file still located inside the MSYS folder, due to the reasons mentioned in previous sections.

IT IS RECOMMENDED TO ADD THE ldpc_jossy FOLDER TO YOUR GROUP'S .gitignore, THIS IS REQUIRED AS EACH TEAM MEMBER WILL BE MAKING SLIGHTLY DIFFERENT CHANGES TO THE ldpc.py FILE AS OUTLINED ABOVE.

The functionality of the package can be confirmed by running `pytest` on the test_ldpc.py file.
This is done in a terminal by running:

```
pytest py/test_ldpc.py
```

Ensure to be in a directory that allows direct access to the py folder in the ldpc_jossy folder, or change the file directory accordingly when calling `pytest`. Note that this requires `pytest` to be installed (run `pip install pytest` if that isn't the case already).

Using the ldpc_jossy package:

The package can be imported into any Python script or IPython Notebook in folders adjacent to the package folder using the command:

```
from ldpc_jossy.py import ldpc
```

Binary lists can now be encoded using the standards outlined in the README.md file (recommended to render this file using Ctrl+K, V).

E.g. encoding a random bit stream:

Build an ldpc.code object, in this case called encoder:

```
encoder = ldpc.code('802.16', '1/2', 54)
```

The encoder only takes `encoder.K` information bits, so ensure your data is only that long!

Encode random bits:

```
x = encoder.encode(random_bits)
```

This can now be fed into your OFDM modulator. Received, demodulated data can be decoded using the following sequence:

This decoding method does not work on purely binary values! Turn your received bitstream into a range that is reasonably outside of ± 1, with negative numbers being binary 1s, and positive numbers being binary 0s. This operates the decoder in hard decision mode. For more advanced decoding techniques, † - See Footnote

Now, the signal can be decoded. Ensure the length of the incoming list is the same as the length of the outgoing bit stream from the encoder!

```
app,it = encoder.decode(y)
```

`app` is the a-posteriori probability that a value in the stream is a 0 (positive entry) or 1 (negative entry). `it` is the number of iterations required to converge on a corrected result. This can be a useful metric to see if more robust codes are required!

The app list can then be turned back into corrected binary lists using:

```
corr = [0 if j > 0 else 1 for j in app]
```

where `corr` will (hopefully) be the input vector into the encoder (in this case `random_bits`). More precisely, this list will be a combination of the data bits (the first `encoder.K` values in this list) and the parity check bits (the rest of `corr`) . To retrieve the data, simply slice `corr` as required. **‡ - See Footnote**

Conclusion:

This whitepaper gave a short overview of the installation and usage of Jossy Sayir's LDPC coding package.

Footnote on Decoding (Max)

† The reason why decoding doesn't work on binary values is because the decoder takes the log likelihood ratio (LLR) between the a priori probabilities as inputs:

$$y_k = Log(\frac{p(x_k = 0)}{p(x_k = 1)})$$

For example, a large negative LLR means there is a high probability that the x sample is a 1, and a small positive LLR means that there is a slightly higher probability that the x sample is a 0 than the x sample being a 1.

This can be used to implement 'soft' decoding as per Jossy's white paper[1] - the LLRs need to be changed to use the distance between constellation points and the real/imaginary axes.

‡ The LDPC decoder returns the a posteriori LLRs, after using the parity information, for all the a priori probabilities inputted. It returns a posteriori probabilities (APPs) for the *whole* input (including parity check bits).

The generator matrix used in LDPC is *systematic* - that is, it has an identity matrix at the start. Hence, the input data is embedded in the start of the codewords. This is why it is okay to discard the extra APPs as the data is found in the first part of the codewords - these are the APPs for the parity check bits.

As an example, consider a simple (6, 3) encoder:

`[0, 1, 0]` - data to be encoded
`[0, 1, 0, 1, 1, 0]` - encoded data
`[5, -4, 1, -7, -6, 5]` - received (a priori) probabilities
`[6, -5, 5, -7, -6, 6]` - APP (a posteriori) probabilities after decoding
`[0, 1, 0]` - after `[0 if j > 0 else 1 for j in app]` and slice

[1] Sayir, J. (2021, May 23). De-mapping QPSK/OFDM for LDPC decoding.

Version history, credits and corrections:

This is Version 1.3 (02/06/2024) of this paper, written by Jeremias Pohle, with a footnote by Max Iyengar.

22/05/2024 - Corrected compiler installation command
02/06/2024 - Added clarification about dealing with the decoded bitstream
02/06/2024 - Added footnotes about LLR probabilities, and why it is okay to discard the second half of the decoded bitstream

Installation instructions are based on the whitepaper by Rory Highnam, James Cozens (https://docs.google.com/document/d/1pLeBU4zk4UMdSok4U0s1pGoBIQ8YzXxj8apGrisRhbs/edit), usage instructions are fully written by Jeremias Pohle