关于使用matlab绘制图像的几个问题。

主要问题：

编写可以将2进制数据，转化为有效数据的m文件。

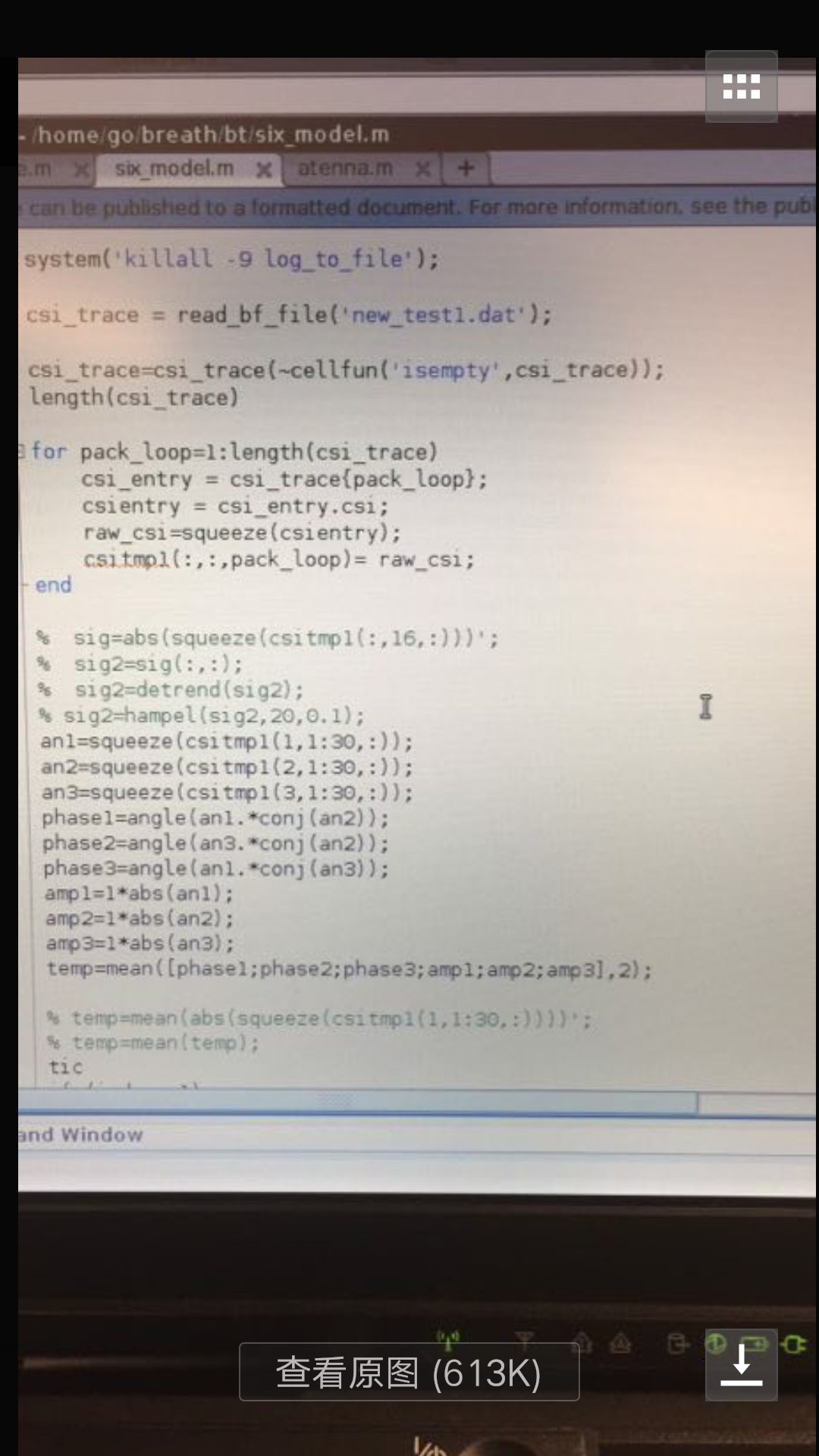
次要问题：

使用有效数据，绘制3维图像。

问题1：

如何将一组csi原始数据转化为amplitude and phase？

方法：



问题1的说明：

原始资料来源：https://dhalperi.github.io/linux-80211n-csitool/faq.html

本实验的一个数据包记录了3分钟内接收的数据，（每0.5秒发送一个数据块，共360个数据块），每个数据块包含3根天线在30个信道接收的数据。

现在的绘图需求是，绘制一个数据包内由1根天线接收的数据的3维图像，x轴为信道，y轴为时间，z轴为amplitude或phase（原z轴为信噪比）。

原建模方式：将原始数据（二进制数据）转化为信噪比，抽取一根天线的的值（30\*360个）作为z轴的值，按照z轴值的顺序，为其设置对应的x，y轴的值，x((1-30)循环360次)，y（（1重复360次）...（30重复360次））。

希望解决的问题：

在原建模方式上，将原始数据分别转化为amplitude或phase，抽取一根天线的的值（30\*360个）作为z轴的值，按照z轴值的顺序，为其设置对应的x，y轴的值，x((1-30)循环360次)，y（（1重复360次）...（30重复360次））。

需求：

编写命令文件（**plot\_csi\_data**.m）

功能：

从文件（sample\_data）中获取原始数据包（csi\_data.dat），按照上述模型（z=amplitude或phase）进行绘图，将yOz图（csi\_data+amplitude+yoz.png）（csi\_data+phase+yoz.png），和xOy图(csi\_data+amplitude+xoy.png)(csi\_data+phase+xoy.png)，保存致文件（csi\_pc）中，按此步骤将文件（sample\_data）中的所有原始数据包（csi\_data.dat）的图像绘制出来保存。

注释：

蓝色标记部分为主要修改点。

### 原绘图方法文档：

### 2. How do I process CSI with MATLAB or Octave? [[−](https://dhalperi.github.io/linux-80211n-csitool/javascript:void(0);)]

#### A. Parsing the CSI trace file

Using MATLAB/Octave, change to the matlab directory in the CSI Tool supplementary material:

cd linux-80211n-csitool-supplementary/matlab

Now read in the CSI trace file. A sample file is included in the supplementary material, but you can also use the file that is generated by following the last step of the [installation instructions](https://dhalperi.github.io/linux-80211n-csitool/installation.html).

csi\_trace = read\_bf\_file('sample\_data/log.all\_csi.6.7.6');

**Note that this uses a MEX-file** compiled from read\_bfee.c to unpack the binary CSI format. If this does not work, recompile this MEX-file using MATLAB/Octave and try again.

#### B. Inspecting the CSI

In our sample file, csi\_trace is a 1x29 cell array, which holds 29 structs. This contains the CSI information for 29 received packets. Let's inspect one of the entries:

>> csi\_entry = csi\_trace{1} **(Note the curly-braces {}, not parentheses ().)**

csi\_entry =

timestamp\_low: 4 **(In the sample trace, timestamp\_low is invalid and always 4.)**

bfee\_count: 72

Nrx: 3

Ntx: 1

rssi\_a: 33

rssi\_b: 37

rssi\_c: 41

noise: -127

agc: 38

perm: [3 2 1]

rate: 256

csi: [1x3x30 double]

Let's break down this display:

* **timestamp\_low** is the low 32 bits of the NIC's 1 MHz clock. It wraps about every 4300 seconds, or 72 minutes. This field was not yet recorded in the sample trace, so all values are arbitrary and always equal 4.
* **bfee\_count** is simply a count of the total number of beamforming measurements that have been recorded by the driver and sent to userspace. The netlink channel between the kernel and userspace is lossy, so these can be used to detect measurements that were dropped in this pipe.
* **Nrx** represents the number of antennas used to receive the packet by this NIC, and **Ntx**represents the number of space/time streams transmitted. In this case, the sender sent a single-stream packet and the receiver used all 3 antennas to receive it.
* **rssi\_a**, **rssi\_b**, and **rssi\_c** correspond to RSSI measured by the receiving NIC at the input to each antenna port. This measurement is made during the packet preamble. This value is in dBrelative to an internal reference; to get the received signal strength in dBm we must combine it with the Automatic Gain Control (AGC) setting (**agc**) in dB and also subtract off a magic constant. This process is explained below.
* **perm** tells us how the NIC permuted the signals from the 3 receive antennas into the 3 RF chains that process the measurements. The sample value of **[3 2 1]** implies that Antenna C was sent to RF Chain A, Antenna B to Chain B, and Antenna A to Chain C. This operation is performed by an antenna selection module in the NIC and generally corresponds to ordering the antennas in decreasing order of RSSI.
* **rate** is the rate at which the packet was sent, in the same format as the rate\_n\_flags defined[above](https://dhalperi.github.io/linux-80211n-csitool/faq.html" \l "faqrate). Note that the antenna bits are omitted, as there is no way for the receiver to know which transmit antennas were used.
* **csi** is the CSI itself, normalized to an internal reference. It is a **Ntx**×**Nrx**×30 3-D matrix where the third dimension is across 30 subcarriers in the OFDM channel. For a 20 MHz-wide channel, these correspond to about half the OFDM subcarriers, and for a 40 MHz-wide channel, this is about one in every 4 subcarriers. Which subcarriers were measured is defined by the[IEEE 802.11n-2009 standard](http://standards.ieee.org/getieee802/download/802.11n-2009.pdf) (in Table 7-25f on page 50).

Now that we've described all the fields of this struct, we need to put them all together to compute the CSI in absolute units, rather than Intel's internal reference level. In particular, we need to combine the RSSI and AGC values together to get RSS in dBm, and include noise to get SNR. If there is no noise, as in the sample case, we instead use a hard-coded noise floor of -92dBm. We use the script get\_scaled\_csi.m to do this:

>> csi = get\_scaled\_csi(csi\_entry);

Finally, **csi** is a 1×3×30 matrix that represents the MIMO channel state for this link. It's units are in linear—i.e., not dB—voltage space. This is the format used in all textbooks I've seen, that is, we've normalized the CSI (in textbooks, usually called *H*) such that there is unit noise.

#### C. Plotting SNR

Let's look at the three different spatial paths on the 1×3 link we measured:

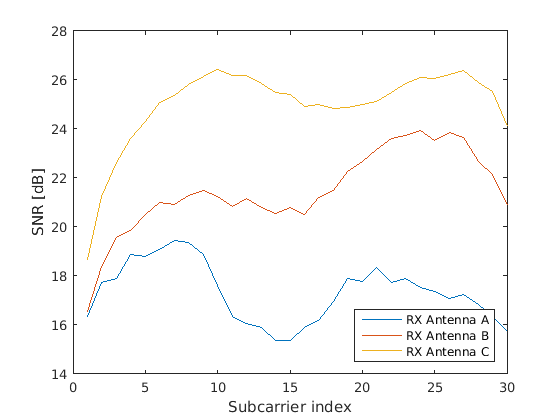
>> plot(db(abs(squeeze(csi).')))

>> legend('RX Antenna A', 'RX Antenna B', 'RX Antenna C', 'Location', 'SouthEast' );

>> xlabel('Subcarrier index');

>> ylabel('SNR [dB]');

In the plot command, squeeze() turns **csi** into a 3×30 matrix by removing the first singleton dimension. db() converts from linear (voltage) space into logarithmic (base-10, power) space. absconverts each complex number into its magnitude. Finally, the **.'** operator transposes the squeezed CSI from 3×30 matrix into a 30×3 matrix, and does not complement the complex numbers. Combined, we get the plot below.

[](https://dhalperi.github.io/linux-80211n-csitool/img/sample_csi_plot.png)

We see that this is a mostly flat link, with relatively little frequency-selective fading (around3 dB for most antenna pairs). However, there is a fair (perhaps 8 dB) difference between the best antenna C and the worst antenna A. This matches the difference between **rssi\_a** and **rssi\_c** (as we expect it should).

#### D. Computing effective SNR values

We'll conclude our discussion of the CSI by showing you how to compute the Effective SNR from our CSI matrices. To do so, we use the get\_eff\_SNRs script, which takes as input a CSI matrix and returns a 7×4 matrix of effective SNR values in linear (power) space. The 4 columns correspond to the effective SNR using the four 802.11 modulation schemes, namely BPSK/QPSK/16QAM/64QAM. The 7 rows correspond to the seven possible antenna selections with 3 antennas and 1, 2, or 3 spatial streams. In particular, the first 3 rows correspond to single-stream transmissions with antenna A, B, or C. The next 3 rows correspond to dual-stream transmissions with antennas AB, AC, or BC. The last row corresponds to a 3-stream transmission using all antennas.

>> db(get\_eff\_SNRs(csi), 'pow')

ans =

22.1821 22.2698 22.9007 24.6297

-156.5356 -156.5356 -156.5356 -156.5356

-156.5356 -156.5356 -156.5356 -156.5356

-156.5356 -156.5356 -156.5356 -156.5356

-156.5356 -156.5356 -156.5356 -156.5356

-156.5356 -156.5356 -156.5356 -156.5356

-156.5356 -156.5356 -156.5356 -156.5356

Okay, that's pretty disappointing! What happened? Well, note that this is a 1×3 link, so the only valid antenna configuration is SIMO with the single transmit antenna we measured. The other 6 rows correspond to a very small SNR, i.e, a large, negative dB.

Let's look at a 3×3 matrix instead:

>> csi\_entry = csi\_trace{20}

csi\_entry =

timestamp\_low: 4

bfee\_count: 91

Nrx: 3

Ntx: 3

rssi\_a: 34

rssi\_b: 39

rssi\_c: 39

noise: -127

agc: 40

perm: [2 3 1]

rate: 272

csi: [3x3x30 double]

>> csi = get\_scaled\_csi(csi\_entry);

>> db(get\_eff\_SNRs(csi), 'pow')

ans =

Inf Inf 32.3435 32.6069

Inf Inf 32.4238 32.6822

Inf Inf 32.2353 32.5051

25.4763 25.5262 25.8974 26.8482

24.6893 24.7490 25.1933 26.5660

21.9185 22.0303 22.8060 24.6483

6.5818 8.2321 12.4185 16.2016

Here, all 7 rows are valid because there are three transmit antennas. We see that all the SIMO streams are very likely to work; in fact, for BPSK and QPSK there are so few errors that MATLAB's error functions can't distinguish it from zero, and the SNR is effectively infinite. The MIMO2 rates are also likely to work, but only some of the MIMO3 rates will work. See our [SIGCOMM 2010 paper](http://homes.cs.washington.edu/~dhalperi/pubs/comm356s-halperin.pdf) for more details.