Google Summer of Code 2020

gr-dpd : Digital Pre-Distortion

Alekh Gupta

March 28, 2020

1 Introduction

Software Defined Radio (SDR)-the ability to process radio signals using software instead of electronics-is undeniably fascinating. GNU Radio provides a wealth of Python functions that you can use to create sophisticated SDR application (or, indeed, any DSP application). For those with little programming knowledge,there is a mostly graphical approach available: GNU Radio Companion, one of the most beautiful GUI softwares available for implementing various SDRs and signal processing systems.
Although it has large no. of signal processing blocks and various OOT(out-of-tree) modules(hosted at CGRAN) available such as: gr-uhd, gr-radar, gr-satellites, gr-gsm, gr-guitar, gr-ham, gr-ofdm,etc. on which I have got my hands so-far. But there is still something which has remained unimplemented as a proper OOT Module,i.e.,a Digital-pre Distorter for PA(power-amplifier) Linearization.

This project mainly focuses on implementing a separate OOT module, namely gr-dpd consisting of some blocks to implement Digital Pre-Distortion Algorithms on a signal to compensate for non-linear responses in the transmitter hardwares,mostly Power Amplifiers.This project will also involve implementing a proper testing tool with GUI for observing the AM-AM and AM-PM responses of an amplifier.
Also as mentioned in project-description,DPD algorithms are not widely available in the Open Source Community.So,this project will also help in fulfilling that void.

1.1 Primary features of the project

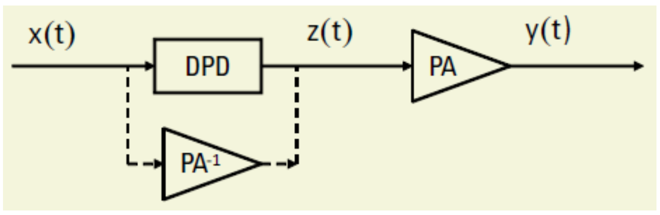
1. Implementation of standard DPD Algorithms and DPD training algorithms for PA linearisation.
2. Integration of the algorithms in blocks of a new module, gr-dpd created using modtool.
3. Write a GUI tool to observe the AM-AM and AM-PM responses of Power Amplifier and design a testbench.
4. Generate the documentation of the module using Doxygen from the documented code.

2 Background Theory

Power amplifiers are essential components in the overall performance and throughput of communication systems, but they are inherently nonlinear. The nonlinearity generates spectral re-growth, which leads to adjacent channel interference and violations of the out-of-band emissions standards mandated by regulatory bodies. It also causes in-band distortion, which degrades the bit-error rate (BER) and data throughput of the communication system.

With newer transmition formats such as wideband code division multiple access (WCDMA) and orthogonal frequency division multiplexing (OFDM, WLAN/3GPP LTE), having high peak-to-average power ratios (PAPR), DPD is one of the most cost-effective linearization techniques.
It features an excellent linearization capability, the ability to preserve overall efficiency, and it takes full advantage of advances in digital signal processors and A/D converters.It provides support for multicarrier signals and high bandwidth.

With the pre-distorter, the power amplifier can be utilized up to its saturation point while still maintaining good linearity, thereby significantly increasing its efficiency.
From Figure below, the DPD can be seen as an "inverse" of the PA. The DPD algorithm needs to model the PA behavior accurately and efficiently for successful DPD deployment.



2.1 DPD Implementation Methods

DPD implementations can be classified into :

Memoryless models and Models with Memory.

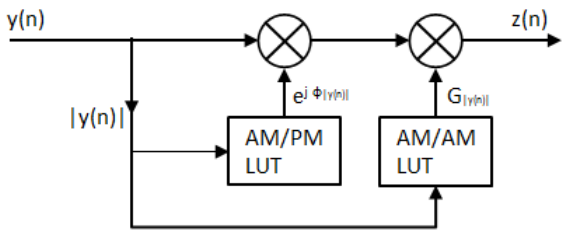
2.1.1 Memoryless models

Memoryless models focus on the power amplifier that has a memoryless nonlinearity, that is, the current output depends only on the current input through a nonlinear mechanism.
Memoryless polynomial algorithm and Look-Up Table (LUT) based algorithms are two key algorithms for memoryless models.
Figure 2 shows the structure of applying the Look-up Table. There are two configurations according to the order of AM/AM and AM/PM.

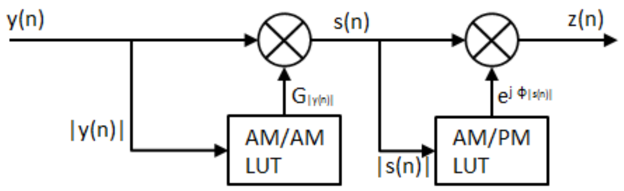
For the first configuration (AM/PM then AM/AM), the input amplitude values for AM/PM LUT and AM/AM LUT are the same. For the second configuration

(AM/AM then AM/PM), the input amplitude values for AM/AM LUT and AM/PM LUT are different.

Configuration 1: AM/PM then AM/AM



Configuration 2: AM/AM then AM/PM



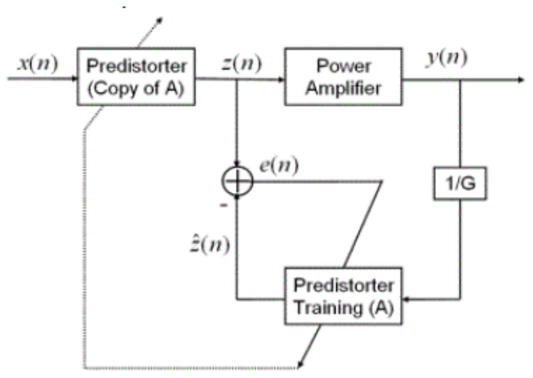
2.1.2 Memory Model

Memory model is commonly used as the signal bandwidth gets wider, such as in WCDMA, mobile WiMAX and 3GPP LTE and LTE-Advanced (up to 100 MHz bandwidth, 5 component carriers of carrier aggregation). For wider bandwidth, power amplifiers begin to exhibit memory effects and the power amplifier becomes a nonlinear system with memory. For such a power amplifier, memoryless pre-distortion can achieve only very limited linearization performance. Therefore, digital pre-distorters must have memory structures.
The most important algorithm for models with memory for Digital pre-distortion implementation is Volterra series and its derivatives.
Volterra's derivatives including Wiener, Hammerstein, Wiener-Hammerstein, parallel Wiener structures, and memory polynomial model are popular in digital pre-distorters. The so-called "memory polynomial" is interpreted as a special case of a generalized Hammerstein model and is further elaborated by combining with the Wiener model.

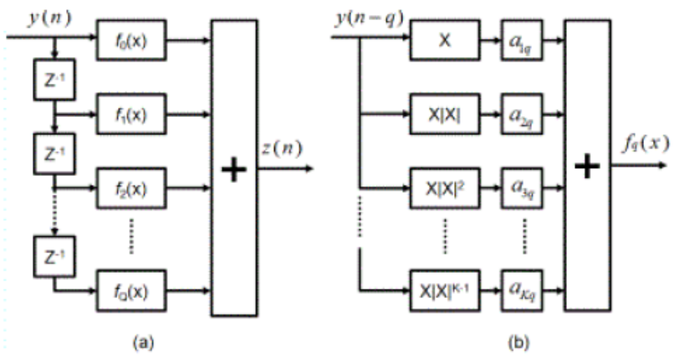
To construct digital pre-distorters with memory structures, there are two types of approaches.
One type of approach is to first identify the power amplifier and then find the inverse of the power amplifier directly. This approach is named as direct learning architecture (DLA). However, obtaining the inverse of a nonlinear system with memory is generally a difficult task.
Another type of approach is to use the indirect learning architecture (IDLA) to design the pre-distorter directly. The advantage of this type of approach is that it

eliminates the need for model assumption and parameter estimation of the power amplifier.

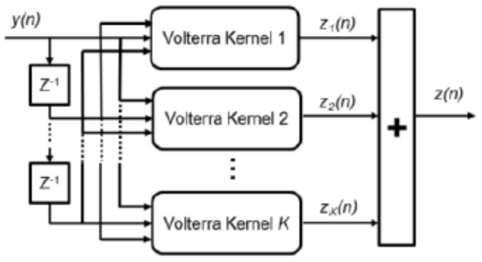
The indirect learning architecture for the digital pre-distorter is shown as following.



The following figure shows the memory polynomial structure. If Q=0, the structure in the following figure becomes memoryless polynomial.



The structure of Volterra series is shown below.



3 Proposed Workflow

Firstly, I will work on implementing the algorithms in raw code(conventional C++) and test them on sample tests.Then, I will create a new separate OOT module namely, *gr-dpd* using the modtool.Then I will add the blocks for each DPD algorithm implemented using the modtool.After the complete integration of each implemented algorithm into blocks according to Block coding guide,I will add required YAML files to make blocks available in GRC. I will keep on documenting my code alongside the development process.
Then, I will implement a testbench including a GUI tool to observe the AM-AM and AM-PM responses of an amplifier. Further, I will work on debugging and improving the algorithm efficiency.
Finally,I will add generate required documentation for the new module *gr-dpd* using Doxygen.

3.1 DPD Algorithm Raw Implementation

There are two general classifications of the DPD algorithms:
Memoryless and Memory based
This project will involve atleast one algorithm implemented from each category.
Digital predistortion and PA behavioral modeling are two areas that are closely related. This is because in order to compensate the distortion introduced by a PA, it is important to find a way to characterize its nonlinear behaviour and the inverse of that behavior. In DPD, this is done with the help of behavioral models.
First I will be implementing the memory based PA modelling.I will be testing these implementations using some random generated vector inputs and their expected outputs according to the PA for computing the success of implementations.

- Memory-Based: For memory based PA behaviour modelling,Volterra series is capable of modelling any non-linear system which has memory effects. However, there is need for truncation of some terms in real-time.
  I will be considering here two most common Volterra-based models:
  - The most commonly known reduced Volterra model is the Memory Polynomial(MP) model, a reduction of the Volterra series in which only products with the same time-shifts are included.The MP model takes the diagonal terms of Volterra series, and its mathematical formulation is given as:

$$y_{MP}(n) = \sum_{k=1}^{K}\sum_{m=0}^{M} a_{km}x(n-m)|x(n-m)|^{k-1}$$

Where x is the input, y is the output, K is the maximum power order, M is the maximum memory depth, and a_{kq} is the kernels (coefficients) of the system.

  - Another important model in this category is the Generalized Memory Polynomial(GMP). This model extends the MP model by also introducing products with different time-shifts, referred as Cross - Terms. The difference between GMP and MP is that, in GMP there are extra terms to compare the terms in MP. The GMP model can be wriiten as:

$$
\begin{aligned}
y(n) = &\sum_{p=1}^{P}\sum_{m=0}^{M-1} a_{pm} u(n-m)|u(n-m)|^{p-1} \\
&+ \sum_{p=2}^{P}\sum_{m=0}^{M-1}\sum_{\substack{g=-m \\ g\neq 0}}^{G} b_{pmg} u(n-m)|u(n-m-g)|^{p-1}
\end{aligned}
$$

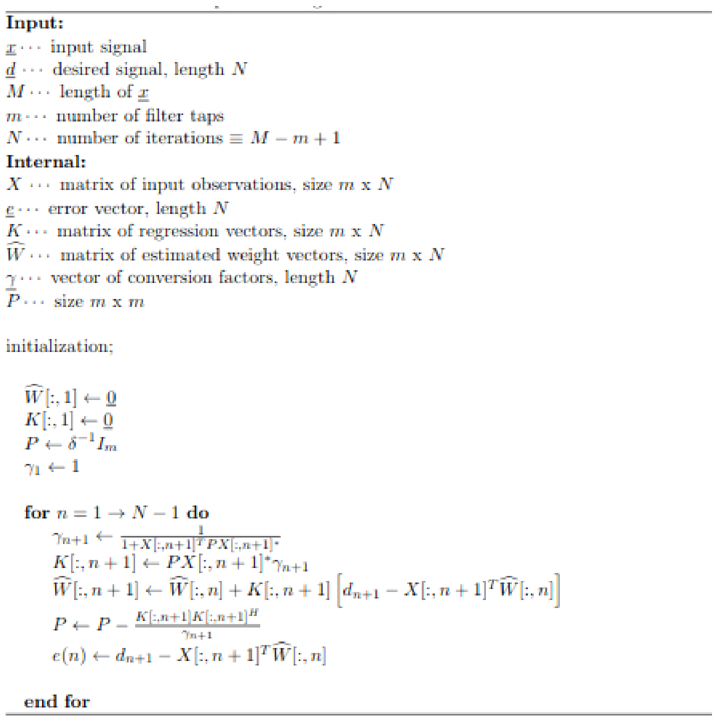
where a_{pm} and b_{pmg} are the model parameters. $|.|$ denotes the absolute value. P, M, and G are the nonlinear order, memory length and cross-term length, respectively. Similar to the Volterra series, the MP and GMP are also linear in the parameters, which means that their parameters can be estimated by least squares techniques.

The proposed Indirect Learning algorithm implementations and coefficient estimation for PA modelling by using MP and GMP models are discussed below.

3.1.1 RLS Algorithm -

Recursive Least Square(RLS) algorithm uses Adaptive Digital-Pre Distortion(ADPD). It is based on modeling nonlinear system (PA and its inverse in this case) by complex valued memory polynomials which are version of Volterra series.

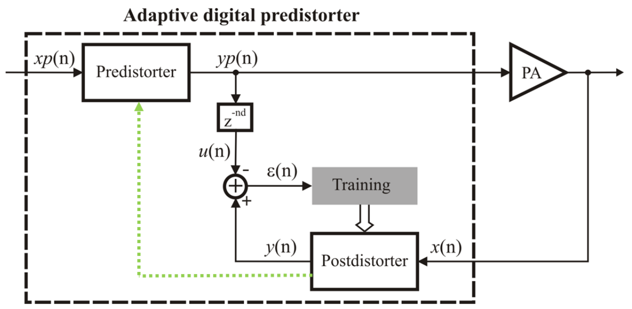
Algorithm[1] :



```
Input:
$\underline{x}$ ⋯ input signal
$\underline{d}$ ⋯ desired signal, length $N$
$M$ ⋯ length of $\underline{x}$
$m$ ⋯ number of filter taps
$N$ ⋯ number of iterations $\equiv M - m + 1$
Internal:
$X$ ⋯ matrix of input observations, size $m$ x $N$
$\underline{e}$ ⋯ error vector, length $N$
$K$ ⋯ matrix of regression vectors, size $m$ x $N$
$\widehat{W}$ ⋯ matrix of estimated weight vectors, size $m$ x $N$
$\underline{\gamma}$ ⋯ vector of conversion factors, length $N$
$P$ ⋯ size $m$ x $m$

initialization;

  $\widehat{W}[:,1] \leftarrow \underline{0}$
  $K[:,1] \leftarrow \underline{0}$
  $P \leftarrow \delta^{-1} I_m$
  $\gamma_1 \leftarrow 1$

  for $n = 1 \rightarrow N-1$ do
    $\gamma_{n+1} \leftarrow \frac{1}{1+X[:,n+1]^T P X[:,n+1]^*}$
    $K[:,n+1] \leftarrow P X[:,n+1]^* \gamma_{n+1}$
    $\widehat{W}[:,n+1] \leftarrow \widehat{W}[:,n] + K[:,n+1]\left[d_{n+1} - X[:,n+1]^T \widehat{W}[:,n]\right]$
    $P \leftarrow P - \frac{K[:,n+1]K[:,n+1]^H}{\gamma_{n+1}}$
    $e(n) \leftarrow d_{n+1} - X[:,n+1]^T \widehat{W}[:,n]$

  end for
```

The notation X[a, b] indicates that the element of the row a and column b of matrix X is selected. Also the colon symbol ":" here means that all the element of the

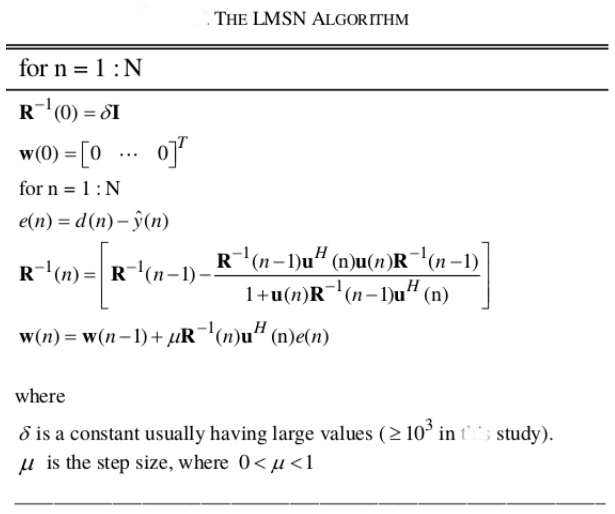
row/column are selected.
Complex valued memory polynomial takes into account both system memory effects as well as the system nonlinearity; in equations N is memory length and M is nonlinearity order.



- Delay line compensates ADPD loop (yp(n) to x(n)) delay
- Postdistorter is trained to be inverse of power amplifier
- Predistorter is simple copy of postdistorter
- When converged (e (n)=0) yp(n)=y(n) => x(n)=xp(n)
- ADPD training algorithm alters complex valued memory polynomial coefficients in order to minimize the difference between yp(n) and y(n), ignoring the delay and gain difference between the two signals.
- Training is based on minimising Recursive Least Square (RLS) E(n) error.

3.1.2 LMSN Algorithm -

The Least-Mean Square Newton(LMSN)[6] algorithm exhibits many desirable characteristics such as stability, robustness and accuracy. An additional advantage of the LMSN algorithm is that it uses less parameters than the RLS algorithm, which utilizes a forgetting factor.
Due to this, the LMSN algorithm is more robust to parameter choices than RLS.


. THE LMSN ALGORITHM

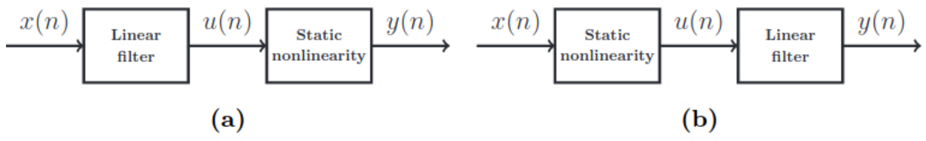
```
for n = 1 : N
$\mathbf{R}^{-1}(0) = \delta\mathbf{I}$
$\mathbf{w}(0) = [0 \ \cdots \ 0]^T$
for n = 1 : N
$e(n) = d(n) - \hat{y}(n)$
$\mathbf{R}^{-1}(n) = \left[ \mathbf{R}^{-1}(n-1) - \frac{\mathbf{R}^{-1}(n-1)\mathbf{u}^H(\mathrm{n})\mathbf{u}(n)\mathbf{R}^{-1}(n-1)}{1+\mathbf{u}(n)\mathbf{R}^{-1}(n-1)\mathbf{u}^H(\mathrm{n})} \right]$
$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu\mathbf{R}^{-1}(n)\mathbf{u}^H(\mathrm{n})e(n)$
```

where

δ is a constant usually having large values ($\geq 10^3$ in t[illegible] study).
μ is the step size, where $0 < \mu < 1$

This version of algorithm removes dependence on user-selected constant by dropping parameter, say α, used in usual Algorithm implementations.

- Memoryless: Now, after memory based, I will come up to the memoryless PA modelling. For memoryless PA behaviour modelling, LUT based model is most basic technique.
  For a given input amplitude, the LUT model indexesthe corresponding AM/AM and AM/PM conversion values using averaging or polynomial fitting and calculates the output as: y(n) =G(|u(n)|)u(n) ,
  where u(n) is input & y(n) is output complex-baseband signal and G(|u(n)|) is the instantaneous complex gain of the PA.



Two-block model structures (a) Wiener model (b) Hammerstein model.

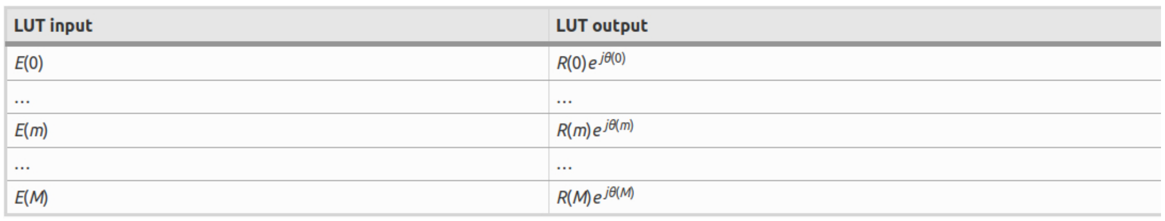
The below direct learning algorithm is based on the LUT based memoryless PA modelling above.

3.1.3 LUT Based Algorithm -

It uses DLD(Direct-learning-architecture). Before performing this Predistortion, a Look-Up Table(LUT) including the information of the amplitude and phase of the predistorted signal should be constructed.

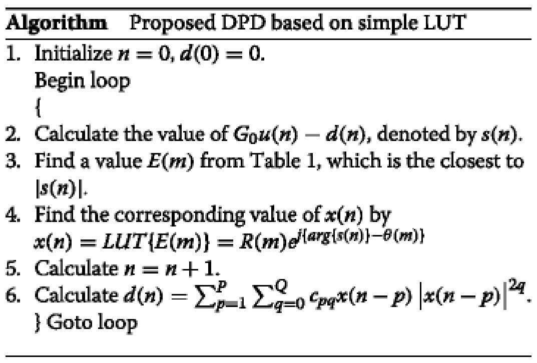
Firstly, the dynamic range of the amplitude |x(n)| of the predistorted signal is estimated according to the maximum amplitude of the original input signal u(n) and the saturation input amplitude of PA. Then the maximum range of |x(n)| is decomposed into M+1 intervals with equal length (the length of each interval is equal to Δ x), denoted by R(m).
Here, R(m)=mΔ x

The LUT is of form given below:



LUT input	**LUT output**
$E(0)$	$R(0)e^{j\theta(0)}$
...	...
$E(m)$	$R(m)e^{j\theta(m)}$
...	...
$E(M)$	$R(M)e^{j\theta(M)}$

After the LUT is constructed, the predistortion algorithm is:



```
Algorithm   Proposed DPD based on simple LUT
1. Initialize $n = 0$, $d(0) = 0$.
   Begin loop
   {
2. Calculate the value of $G_0 u(n) - d(n)$, denoted by $s(n)$.
3. Find a value $E(m)$ from Table 1, which is the closest to
   $|s(n)|$.
4. Find the corresponding value of $x(n)$ by
   $x(n) = LUT\{E(m)\} = R(m)e^{j\{\arg\{s(n)\}-\theta(m)\}}$
5. Calculate $n = n + 1$.
6. Calculate $d(n) = \sum_{p=1}^{P} \sum_{q=0}^{Q} c_{pq} x(n-p) \left| x(n-p) \right|^{2q}$.
   } Goto loop
```

The proposed approach[4] based on a simple LUT can significantly improve the time efficiency of DPD process than the corresponding root-finding approach, but its linearization performance is proportional to the table size. To obtain good linearization performance, the table size must be sufficiently large. Therefore, some interpolation techniques should be introduced in the simple LUT. The linear interpolation and the quadratic interpolation are adopted in LUT, respectively.

To test above Raw implementations, I will use vector lists of random numbers as input signals and then compare the output vectors with expected output from a non-linear PA for that input vector.

3.2 OOT Module gr-dpd

For making the implemented DPD algorithms available for direct use in flowgraphs, there is a need to create a new OOT Module namely, gr-dpd. It can be easily created using the modtool as:

```
        gr_modtool newmod gr-dpd
```

Then, to make algorithms available for use in flowgraphs we need blocks to implement each of them which can be added to the the gr-dpd using the modtool:

```
       gr_modtool add -t general -l cpp stream_to_vec
```

Some of the basic Blocks which are essential prior to and after the distortion process are:

- *stream-to-vec*: This block will convert the input signal stream to the required column vector form for further signal processing.
  Sample implementation for this block's *work* function would look like:

```
int streams_to_vec_impl::work(int noutput_items,
          gr_vector_const_void_star&input_items,
          gr_vector_void_star& output_items)
{
    size_t itemsize = input_signature()->sizeof_stream_item(0);
    int nstreams = input_items.size();
    const char** inv = (const char**)&input_items[0];
    char* out = (char*)output_items[0];

    for (int i = 0; i < noutput_items; i++) {
      for (int j = 0; j < nstreams; j++) {
         memcpy(out, inv[j], itemsize);
         inv[j] += itemsize;
         out += itemsize;
      }
    }
    return noutput_items;
}
```

Another implementation can be using Armadillo library by Srikanth Pagadarai as here.

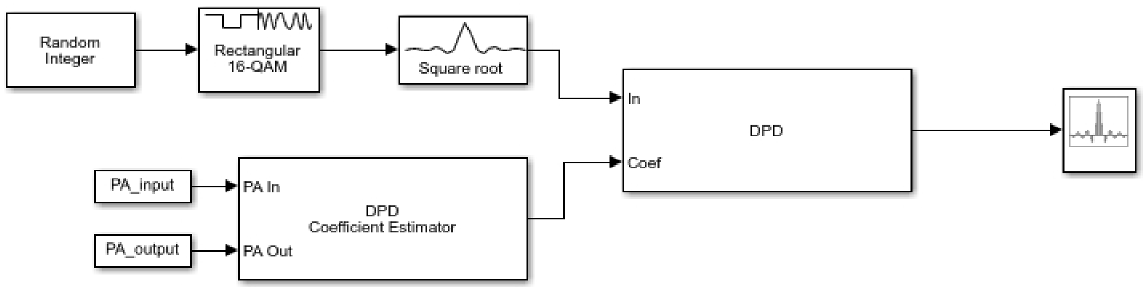
- power-amplifier: This block is essential for feeding the predistorted signals into and simulation of flowgraphs to test the various blocks defined in the gr-dpd module to implement the algorithms.

  An implementation for the Power-Amplifier based on the Weiner model as a Heirarchial block is given here.
  Another implementation based on a GMP model will be considered for proper testing of algorithms implemented.

  Am embedded python block code will look as:

```

def__init__(self,Order1=np.complex(5,0),Order3=np.complex(0,0),
    Order5=np.complex(0,0),Order7=np.complex(0,0),Order9=np.complex(0,0)):
 # only default arguments here
 """arguments to this function show up as parameters in GRC"""
          gr.sync_block.__init__(self,name="Wiener Power
              Amplifier",
          # will show up in GRC
          in_sig=[np.complex64],
          out_sig=[np.complex64]
          )
          # if an attribute with the same name as a parameter is
              found
          self.Order1 = Order1
          self.Order3 = Order3
          self.Order5 = Order5
          self.Order7 = Order7
          self.Order9 = Order9

def work(self, input_items, output_items):
       """evaluate the non-linear amplification by odd order
           coefficients"""
          output_items[0][:] = input_items[0]*self.Order1+
          ((input_items[0])**3)*self.Order3+
          ((input_items[0])**5)*self.Order5+
          ((input_items[0])**7)*self.Order7+
          ((input_items[0])**9)*self.Order9

          return len(output_items[0])
```

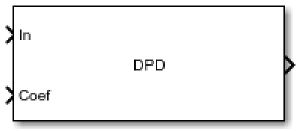
A general flowgraph which depicts the blocks essential for DPD implementation is as:



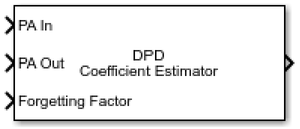
Further, Blocks which are to be added for Pre-Distortion Algorithm implementations are:

- pre-distorter: This block will be perform pre-distortion training(RLS) on the PA input and sends the PA input to Post-distorter and in beginning send zeros to the output.It's *work* function will look like:

```
int predistorter_training_impl::work(int noutput_items,
gr_vector_const_void_star &input_items,
gr_vector_void_star &output_items)
{
   gr_complex *out = (gr_complex *) output_items[0];
   // Do <+signal processing+>
   predistorter_training_colvec = d_predistorter_training_colvec;
   update_predistorter_training = d_update_predistorter_training;
   for (int item = 0; item < noutput_items; item++)
 {
  // get PA input which has been arranged in a vector format
  cx_fmat yy_cx_rowvec( ((gr_complex *) input_items[0])+item*d_M,
      1, d_M, COPY_MEM);

  // get number of samples consumed since the beginning of time by
      this block from port 0
  nread = this->nitems_read(0);

  // send the zeros at the beginning and samples without any
      predistortion
  //apply predistortion and send the PA input to postdistorter
    if (nread >= nskip)
   {
   out[item]=as_scalar(conv_to<cx_fmat>::from(yy_cx_rowvec*predistorter_training_colvec)
       );
   }
    else
    out[item] = *(((gr_complex *) input_items[0])+item*d_M);
  }
}
```

It's Analogy in above flowgraph is: 

- post-distorter: This block is useful in updating the Iteration No., estimating Weight vector to send it to pre-distorter.This block is also used in updating time for factor 1/sqrt(gamma) applied at PA output.

Analogous block in flowgraph is: 

- signum-funct: This block implements the standard Sigma function required for while Signal-Processing during Predistortion. It's work function looks as:

```
int signum_func_impl::work(int noutput_items,
gr_vector_const_void_star &input_items,
gr_vector_void_star &output_items)
{
   const float *in = (const float *) input_items[0];
   float *out = (float *) output_items[0];

   // Do <+signal processing+>
   for (int i = 0; i < noutput_items; i++)
     out[i] = ( in[i] > 0.0 ) ? 1.0 : -1.0;
   // Tell runtime system how many output items we produced.
      return noutput_items;
}
```

- vec-vec-mul: This block implements the multiplication of column vectors during Signal Processing. The below implementation using Armadillo library is:

```
int vector_vector_multiply_impl::work(int noutput_items,
       gr_vector_const_void_star &input_items,
       gr_vector_void_star &output_items)
{
    gr_complex *out = (gr_complex *) output_items[0];
    // Do <+signal processing+>
    for (int item = 0; item < noutput_items; item++)
    {
      cx_fcolvec input_vec( (gr_complex *)
          input_items[0]+item*d_vec_len,
                          d_vec_len,COPY_MEM,FIX_SIZE );
      out[item] = as_scalar(arma_const_vec*input_vec);
    }
      return noutput_items;
}
```

- lmsn-pre-distoter: This block will be used to implement pre-distortion as an alternate to pre-distorter to implement the lmns algorithm.It will eliminate the forgetting factor.

- lut-generator: This block will be used to generate the LUT including the information of the amplitude and phase of the predistorted signal in form of Map or Matrix.
- lut-pre-distorter: This block will take input of the LUT stored in form of Map or Matrix. And perform signal processing based on LUT algorithm using LUT values.
  Some other blocks to perform intermediate Signal Processing and form Overall Output Vector will be defined too in gr-dpd.

3.3 Test Code and YAML files

For proper testing of each block and ensuring proper implementation of Pre-distortion Algorithm in flowgraph, it is essential to include unittests for each block.
modtool already provides QA(quality assurance) test code files as say, qa-signum-func.py in python directory of gr-dpd. An example testcode to test signum-func block is as:

```python
def test_001_signum_func(self):
       srcd = (1,2,18,-4,-1)
       expected = (1.0,1.0,1.0,-1.0,-1.0)
       src = blocks.vector_source_f(srcd)
       sqr = dpd.signum_func()
       dst = blocks.vector_sink_f()
       self.tb.connect(src, sqr)
       self.tb.connect(sqr, dst)
       self.tb.run()
       result = dst.data()
       self.assertFloatTuplesAlmostEqual(expected, result, 5)
```

Testing of each block using "make test" will be done and QA tests will be added. For testing using hardware, there will be use of gr-uhd module. I currently have a portable FM RTL2832U + R820T2 reciever. I was planning to have a USRP B210 if I get selected. This seems to be a better choice to me while mentor's view will be considered before buying.As USRP are widely used, they are better choice for testing purposes.
Now, to make these blocks available in the GRC of gnuradio-maint-3.8, YAML files have to be added.There are two ways for this: First, Adding the XML files using modtool as:

```
   gr_modtool makexml pre_distorter
```

Then using the converter provided by GNU Radio(will do 95% conversion) as(from root of module):

```
   gr_modtool update --complete
```

Second way is writing the YAML GRC bindings by myself which I can do so easily. After that these blocks can be used in GRC after doing "make install" from build directory.

3.4 GUI Tool for AM-AM and AM-PM responses:

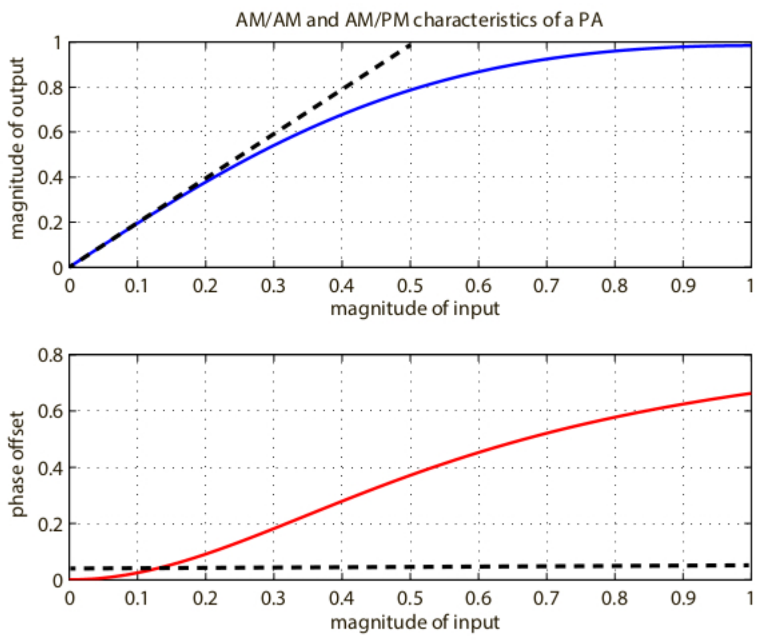
The design of power amplifiers entails a critical trade-off between power efficiency and linearity.The power efficiency of a PA can be defined as its ability to convert the DC power of the supply in output power.
The behaviour when the PA is operating next to saturation becomes nonlinear and the gain starts being dependent on the input. Conventionally, memoryless nonlinearities can be characterized by the AM/AM and AM/PM responses of the PA, in which output amplitude and phase offset are given as functions of the current input amplitude.

My thoughts on AM-AM and AM-PM plots and their use:

AM-AM and AM-PM plots are most important and traditional method for characterisation of Power Amplifiers.The AM/AM distortion is the amplitude distortion as a function of the amplitude of an input signal and AM/PM is the phase distortion as a function of the same amplitude.They help in determining the amount of distortion that the non-ideal PA causes. Amplitude-Amplitude(AM-AM) Distortion shows the Gain-offset while Amplitude-to-Phase(AM-PM) Distortion specifies the Phase Deviation.By knowing the gain offset (AM/AM) and phase distortion (AM/PM) caused by the PA for particular sample at instant (n),the pre-distorter can be configured to equalize the undesired PA gain and compensate for the undesired phase shift.

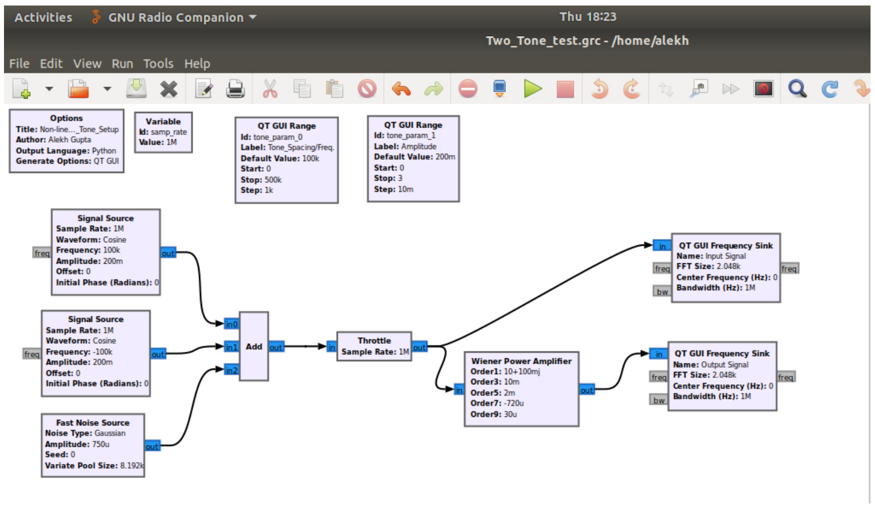
So, finally , I will add a GUI tool similar to Qtgui-sink in GRC.This will involve adding blocks to implement the signal processing and display widget for each of the two plot characteristics.

- For AM-PM plot, there will adding of two blocks namely,*phase-sink* and *phase-plot* for both computing the phase offset or degree of output signal from desired signal and then displaying the plot using Qt widgets.
- For AM-AM plot, there will be adding of two new blocks to gr-qtgui namely,*amplitude-sink* and *amplitude-display* for real time-plot.
- After, this there will be adding of a new final block namely, dpd-sink to form a separate qtgui widget which would invoke above blocks to plot both AM-AM and AM-PM characteristics in same Widget Window in different tabs for a better visualtion of both characteristics of PA output.



[5] This shows Characteristics of a PA besides desired characteristics.

- A testbench can be easily implemented in form of flowgraphs in GRC to ensure proper functioning of blocks and their combination to perform pre-distortion. One such Two-Tone Setup Flowgraph[2] is given below.Here Power Amplifier is an Embedded Python Block implementing a Non-linear PA with Odd power terms, discussed above.



Options
Title: Non-line..._Tone_Setup
Author: Alekh Gupta
Output Language: Python
Generate Options: QT GUI

Variable
Id: samp_rate
Value: 1M

QT GUI Range
Id: tone_param_0
Label: Tone_Spacing/Freq.
Default Value: 100k
Start: 0
Stop: 500k
Step: 1k

QT GUI Range
Id: tone_param_1
Label: Amplitude
Default Value: 200m
Start: 0
Stop: 3
Step: 10m

Signal Source
Sample Rate: 1M
Waveform: Cosine
Frequency: 100k
Amplitude: 200m
Offset: 0
Initial Phase (Radians): 0

Signal Source
Sample Rate: 1M
Waveform: Cosine
Frequency: -100k
Amplitude: 200m
Offset: 0
Initial Phase (Radians): 0

Fast Noise Source
Noise Type: Gaussian
Amplitude: 750u
Seed: 0
Variate Pool Size: 8.192k

Throttle
Sample Rate: 1M

Wiener Power Amplifier
Order1: 10+100mj
Order3: 10m
Order5: 2m
Order7: -720u
Order9: 30u

QT GUI Frequency Sink
Name: Input Signal
FFT Size: 2.048k
Center Frequency (Hz): 0
Bandwidth (Hz): 1M

QT GUI Frequency Sink
Name: Output Signal
FFT Size: 2.048k
Center Frequency (Hz): 0
Bandwidth (Hz): 1M

3.5 Documentation

Finally to make gr-dpd accessible to all potential users, there must be proper documentation of the module. I will use Doxygen to generate the documentation for the gr-dpd from the already documented code. This involves:

- Documenting the code.
- Creating a configuration file. It's template can be created as:

```
doxygen -g <config-file>
```

- Running the doxygen to generate the documentation(HTML or LaTeX based) as:

```
doxygen <config-file>
```

4 Timeline

During Community bonding period, I will interact with my Mentor and other community members to get myself known to the Community. I will also go through the source code again thouroughly to get myself well versed with the coding style in gnuradio.I will also work on determining more better ways to implement the DPD algorithms in gr-dpd.This will ensure my contributions are at-par with the community standards.I will discuss complete details of my development plan with the mentor to ensure smooth coding period.

I will keep documenting my code alonside the development only. I am having vacations in months of June, July and half month of May. So, I can devote my full-time during these days. I will be able to work around 40-42 hours a week during this period while I will be able to work for around 36-38 hours a week during August month. I will keep my Deliverables on a weekly-basis as I am having a 13-week coding-period.

Whether I have any plans for this summer?
No, I have considered GSoC as my "top-most" priority as I have not applied for any internship and am totally reliant on the GSoC with gnuradio for this summer.

From where I will work for the GNU Radio this GSoC?
I will be working from my permanent home in Nahan, Himachal Pradesh(India) with no expected movements outside.

The expected timeline for my project is given below:

Timeline of the project

May 4 - May 31	Make a detailed Development Plan about gr-dpd module and its architecture, and do some sample DPD algorithm implementations.
June 1 - June 4	Implement RLS Algorithm in raw form alongwith the memorybased PA model and prepare sample tests.
June 5 - June 8	Implement LMNS Algorithm in raw form and do sample testing using the PA model based on memory.
June 9 - June 14	Implement methods to generate Look-Up Table and then implement the LUT based Algorithm in raw form alonside testing with PA memoryless model implemented.
June 15 - June 20	Creating gr-dpd with modtool and adding the blocks as stream-to-vec, signum-func.
June 20 - June 24	Adding the block pre-distorter, its test code for RLS Algorithm and docs in block code.
June 25 - June 27	Adding the post-distorter block, its test code and docs in code.
June 28 - July 4	Adding the vec-vec-mul block,other blocks for signal processing in algorithms as required and their test codes and docs.
July 5 - July 7	Testing the RLS Algorithm using a testbench flowgraph.
July 8 - July 12	Adding the lmns-pre-distorter block and its test code.
July 13- July 19	Adding changes for LMNS algorithm in blocks and testing it with a flowgraph.
July 20 - July 26	Adding the lut-generator block and lut-pre-distorter block and document their code.
July 27 - Aug 2	Testing LUT based algorithm with test codes and flowgraph.And add the required YAML files for all blocks.
Aug 2 - Aug 12	Writing a GUI tool using QtWidgets to observe AM-AM and AM-PM responses.
Aug 12 - Aug 20	Implementing a proper set of tests using flowgraphs in python code for all the blocks and all the algorithms implemented.
Aug 20 - Aug 24	Generating the documentation for gr-dpd using Doxygen from the already documented code. Work on finalising the code and removing any bugs.
Aug 24 - Aug 31	Complete the project to make it user ready and submit the final work product.

5 Deliverables of GSoC 2020

The deliverables of the GSoC project are as follows:

- Proper implementation of DPD Algorithms in the form of blocks of OOT module gr-dpd to be functional in a flowgraph implementation.
- Proper testing and debugging of the DPD Algorithm blocks using test codes as well as flowgraphs. Making the blocks available in GRC.
- A nicely implemented GUI tool similar to Qtgui-sink to view AM-AM and AM-PM responses of an amplifier.
- Making gr-dpd easily user acessible by proper documentation of API as well as blocks using Doxygen.

5.1 Milestones

- Phase-1: DPD Algorithms implemented in raw code and test cases prepared for each algorithm.
- Phase-2: All the algorithms implemented in form of blocks of gr-dpd available for use in GRC in the flowgraphs.Adding the testbenches for each block and each algorithm to be implemented.
- Final Evaluation: A GUI tool implemented using Qtgui to observe the AM-AM and AM-PM responses of an amplifer. Documentation for gr-dpd generated with Doxygen from already documented code itself. Completion of testing of blocks and algorithm implementation using flowgraphs.

5.2 Review/Merge Cycle

The code review can occur alongside the development process of various blocks and implementations.While merging of the code into repository for the module gr-dpd will occur as following:

- June 28: Merging of the code for some blocks implemented such as stream-to-vec,power-amplifier, pre-distorter, postdistorter.
- July 26: Merging the code for blocks: lmsn-pre-distorter, lut-generator, lut-predistorter and other blocks for intermediate processing.
- August 20: Merging of code for GUI tool into gnuradio module gr-qtgui for adding functionality of AM-AM and AM-PM responses.And also QA test codes and testbenches involving flowgraphs.
- August 26: Merging of final product module gr-dpd available for direct use in GRC, with final documentation generated from regularly documented code.

5.3 Testing Work

Testing alongside is quite essential for successful implementation of DPD algorithms in form of gr-dpd blocks. I will keep on testing the blocks, added in gr-dpd with python QA code as well as using flowgraphs. This will ensure smooth workflow. Proper testbenches will be added for testing of gr-dpd blocks and algorithms implemented with them.I will also use a USRP B210 SDR for ensuring proper testing using the gr-uhd module.

6 Acknowledgement

I have thoroughly gone through the GSoC StudentInfo page and GSoC Manifest page. I also accept the three strikes rule and the details mentioned.I hereby assure that I will abide by the rules and regulations. I will follow the community guidelines and etiquettes while interaction with any of the member or mentors.
I also assure that I will communicate with the assigned mentor regularly, maintain thorough transparency and keep my work up to date.

7 License

The entire code during the coding period will be transparent, i.e., available on Github. The code submitted will be GPLv3 licensed.

8 Personal Details and Experience

I am a second year undergraduate at National Institute of Technology, Hamirpur. My areas of interest are Digital-Signal Processing, algorithm analysis, design and competitive programming. I am proficient in C, C++, Python, C# and HTML/CSS.I am quite experienced in using Git as I have been part of organising various workshops on Basic Git and VCS in our college for first-year students.
I have only small contributions to Open-Source since I am a beginner. I have actually started to make some really worth contributions only with GNU radio since "Cyberspectrum is the best spectrum".It will prove to be a great opportunity if I am selected for this GSoC project as well as my first experience in working with some really big organisation.I will not get any extra credits for the GSoC project. I am proficient in two human languages,i.e., English and Hindi.

I have the experience of working closely with a team as I am an executive member of Team .EXE at NIT Hamirpur, the departmental team of C.S.E. Department at NIT Hamirpur.My major project as a part of the group is EXEPlore, a gaming platform(using Django) for organising an online Game Event among the college students during the technical fest of our college.It consisted of four games made using C# and Unity Engine.I also developed one of the game for this event "Tesla Car Race" using the Unity Engine. It involved C# scripting based on vector-physics principles.I was also problem setter for the first CTF event of our college NITH-CTF 1.0.

I am also a member of GNU Linux Users Group which is aimed at spreading the culture and craze of Open-Source among people both in and outside NIT Hamirpur by organizing contests, workshops, delivering lectures, etc. We also organise "SFD (Software Freedom Day)" in our college involving guest talks by eminent personalities like once Richard Stallman himself visited the GLUG-NITH.

I started off with GNU Radio in December 2019.I have done some flowgraph projects using GRC in beginning.To get familiarized with the code, I made the following contributions to the codebase:

1. Pull request # 3176: grc: Added new menu option under 'Help' for Keyboard-Shortcuts.
2. Pull request # 3258: gr-utils: '#'replaced by '%' for commenting in codes.
3. Pull request # 3263: modtool: Replaced str.format() by Python f'strings in all codes

I will always be available on email or Google Hangouts or Skype for any kind of discussion or query.
I have also got my CLA process completed. I am highly interested to contribute to GNU Radio after the GSoC period. After the period, I'll mainly focus on debugging the gr-dpd module and try to get it merged into the main source tree of GNU Radio. I'll always be available for fixing the bugs that come up in the gr-dpd.
Some of the couses relevent taken by me so far, which are relevant to the project are:

- CSD-115 Basic Electronics Engineering (Transistors,FETs,Amplifiers, OpAmps) by Dr. CB Kushwah
- CSD-118 Electronics Lab
- CSD-125 Basic Electrical Engineering
- CSD-214 Microprocessor and Interfacing (8085,8086,8259,8251,etc. Architectures)
- CSD-215 Digital Electronics & Logic Design (Hamming Codes, Combinational logics, Sequenctial logic and Flip-Flops.)
- CSD-218 Microprocesser Lab
- CSD-219 Digital Electronics Lab
- CSD-311 Modelling and Simulation.

I have done various projects and algorithm implementations in C++ and Python. Some of my relevant projects are:

- Web Server: It is a very simple, fast, multithreaded, platform independent HTTP and HTTPS server and client library implemented using C++11 and Asio (both Boost.Asio and standalone Asio can be used).
- Paint App.:This is a clone of windows paint using PyQt.
- Calculator: It is a simple GUI application with two versions:one in PyQt and other using C++ Qt-Creator.

- 2048 Game: It is the popular 2048 game implemented using tkinter and pygame python libraries.
  Apart from these, I have done various implementations of Algorithms and Data Strutures in C++.

Here is a link for my Resume.

8.1 Other Details

Address	:	Nahan, Himachal, India
Email	:	alekhgupta1441@gmail.com
Github	:	https://github.com/alekhgupta1441/
LinkedIn	:	https://www.linkedin.com/in/alekh-gupta-890400171/
Codechef	:	https://www.codechef.com/users/alekhgupta1441

8.2 Other Links

- https://github.com/alekhgupta1441/GSoC-Proposal
- https://github.com/alekhgupta1441/GRC-Works

9 Conclusion

Digital Pre-Distortion (DPD) is one of the most fundamental building blocks in wireless communication systems today. It is used to increase the efficiency of Power Amplifiers. By reducing the distortion created by running Power Amplifiers in their non-linear regions, Power Amplifiers can be made to be far more efficient. So, gr-dpd will prove to be a great module for useful for various purposes ranging from testing to simulations. Along with a GUI tool for AM-AM and AM-PM responses and proper documentation, it will be easily accessible for use by students,researchers,etc.Again I will say "Cyberspectrum is the best spectrum".

10 References

1. https://publik.tuwien.ac.at/files/PubDat-242282.pdf
2. https://www.derekkozel.com/talks/FOSDEM2019-Digital-PreDistortion.pdf
3. https://github.com/SrikanthPagadarai/gr-dpd
4. https://link.springer.com/article/10.1186/s13638-016-0628-y
5. http://www.summittechmedia.com/highfreqelec/Sep04/HFE0904-Nezami.pdf
6. https://www.ieice.org/nolta/symposium/archive/2014/articles/D1L-B1-6093.pdf