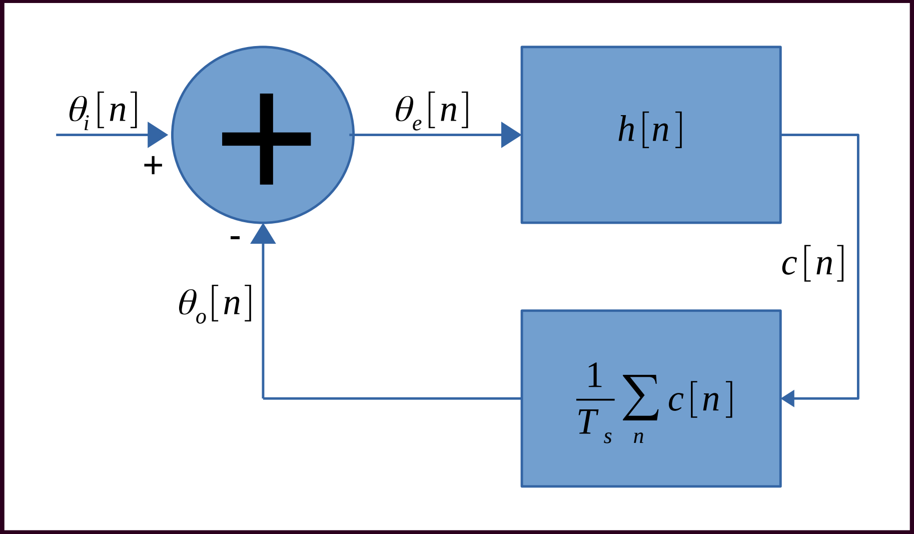
PLLpy

PLLs in SDR Softwares

Dr. Selmeczi János
HA5FT
ha5ft@freemail.hu



PLLpy

What is pllpy?

```
class EdgePhaseDetector :
    def __init__(self, threshold) :
        self.threshold = threshold
        self.I = 0.0

    def next(self, I, Q, gain=1.0) :
        if (abs(Q) <= self.threshold) :
            sQ = 0
        elif Q > 0 :
            sQ = 1
        else :
            sQ = -1

        if (abs(self.I) <= self.threshold) :
            sIb = sQ
        elif self.I > 0 :
            sIb = 1
        else :
            sIb = -1

        if (abs(I) <= self.threshold) :
            sIf = -sQ
        elif I > 0 :
            sIf = 1
        else :
            sIf = -1

        if (sIf != sIb) :
            e = -sIb * Q
        else :
            e = 0
        self.I = I

        e *= math.pi # to correct the detector gain

        return e
```

Pll test bench software

- Open source
- Written in python
- Published on github https://github.com/ha5ft/pllpy

It's major goals

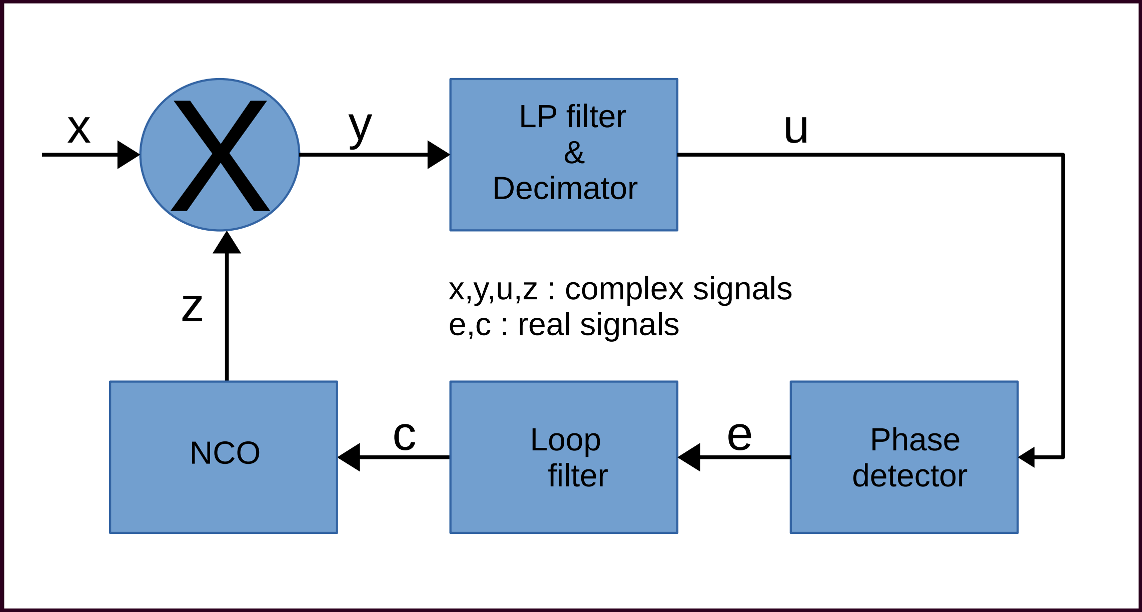
- Publishes important algorithms used to implement PLLs
- Enables its users to learn how software PLLs are working
- Makes possible the easy experimenting with various PLLs
- Enables users to try algoritms before implementing them in target systems

Most of the algorithms published have been implemented ih FPGA code

PLLpy

What is a PLL?

Control loop for tracking the phase of a complex sinusoidal signal



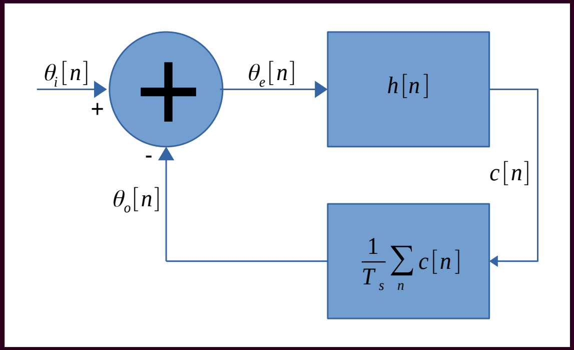
Components:

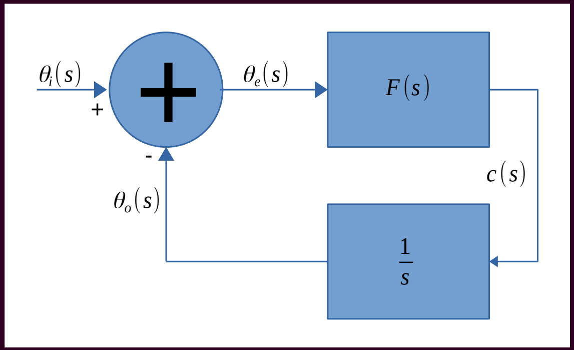
- Complex mixer
- Low pass filter & optional decimator
- Phase detector
- Loop filter
- NCO

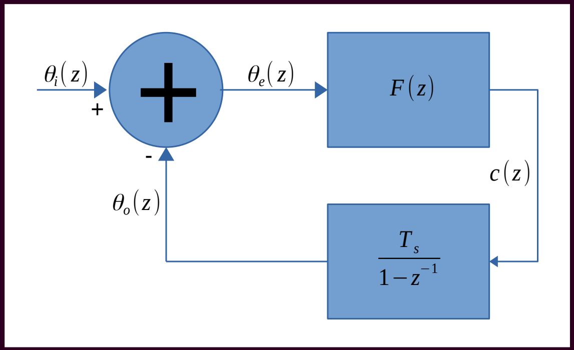
PLLs are nonlinear, the phase is burried in complex exponential function. The phase detctor is nonlinear too.

PLLpy Linear models of a PLL

- Linear models for various domains
  - Discrete time
  - Laplace transformation (s)
  - Z transformation (z)
- Linearization
  - Complex exponential replaced by phase
  - Mixer, LP filter and phase detector replaced by an adder substractor
  - NCO is replaced by an integrator







These linear modells are used for most of the PLL analysis

PLLpy

Transfer functions

Open loop:

$$G(s)=\frac{\theta_o(s)}{\theta_e(s)}=\frac{F(s)}{s}$$

$$G(z)=\frac{\theta_o(z)}{\theta_e(z)}=\frac{T_s F(z)}{1-z^{-1}}$$

Sytem:

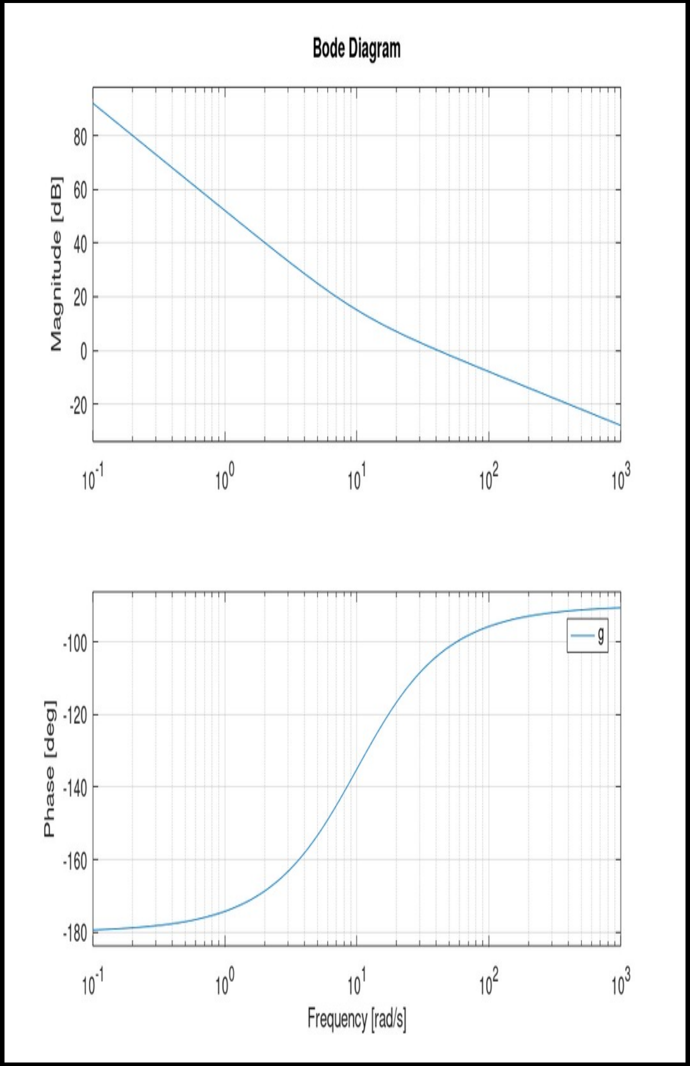
$$H(s)=\frac{\theta_o(s)}{\theta_i(s)}=\frac{G(s)}{1+G(s)}$$

$$H(z)=\frac{\theta_o(z)}{\theta_i(z)}=\frac{G(z)}{1+G(z)}$$

Error:

$$E(s)=\frac{\theta_e(s)}{\theta_i(s)}=1-H(s)$$

$$E(z)=\frac{\theta_z(s)}{\theta_i(z)}=1-H(z)$$



G(s), G(z) contains and additional integrator reprezenting the NCO

PLLpy

Loop filters

Type 1 : 1 integrator in H(s), H(z)

$$F(s)=K_p$$

$$F(z)=K_p$$

Used when phase synchronizatos needed only

Type 2 : 2 integrator in H(s), H(z)

$$F(s)=K_p\left(1+\frac{\omega_0}{s}\right)$$

$$F(z)=K_p\left(1+\frac{\omega_0 T_s}{1-z_{-1}}\right)$$

Type 3 : 3 integrator in H(s), H(z)

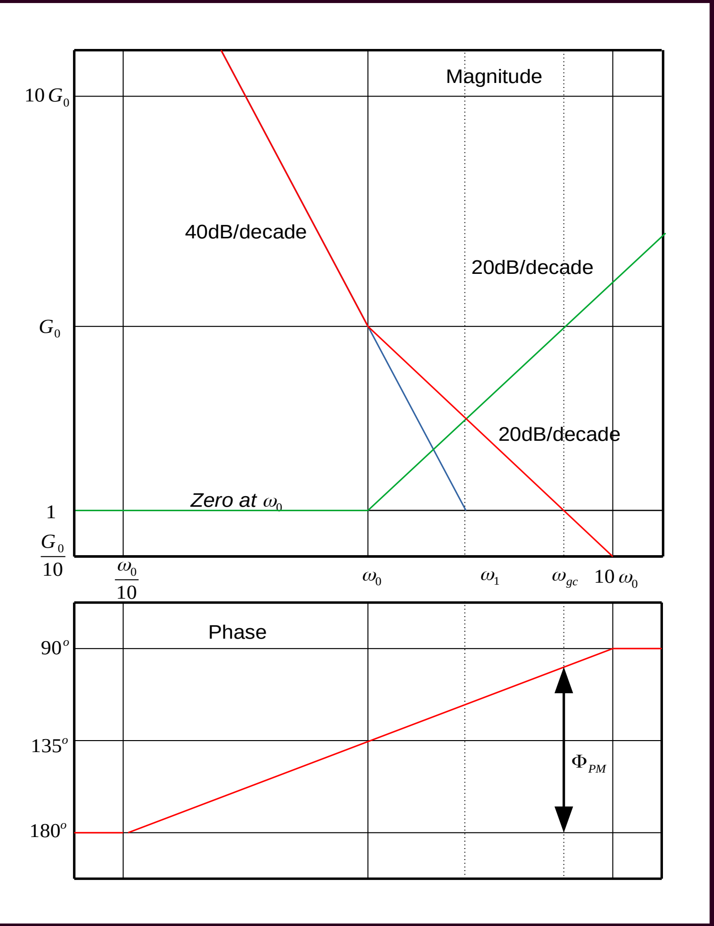
$$F(s)=K_p\left(1+\frac{\omega_0}{s}\right)^2$$

$$F(z)=K_p\left(1+\frac{\omega_0 T_s}{1-z_{-1}}\right)^2$$

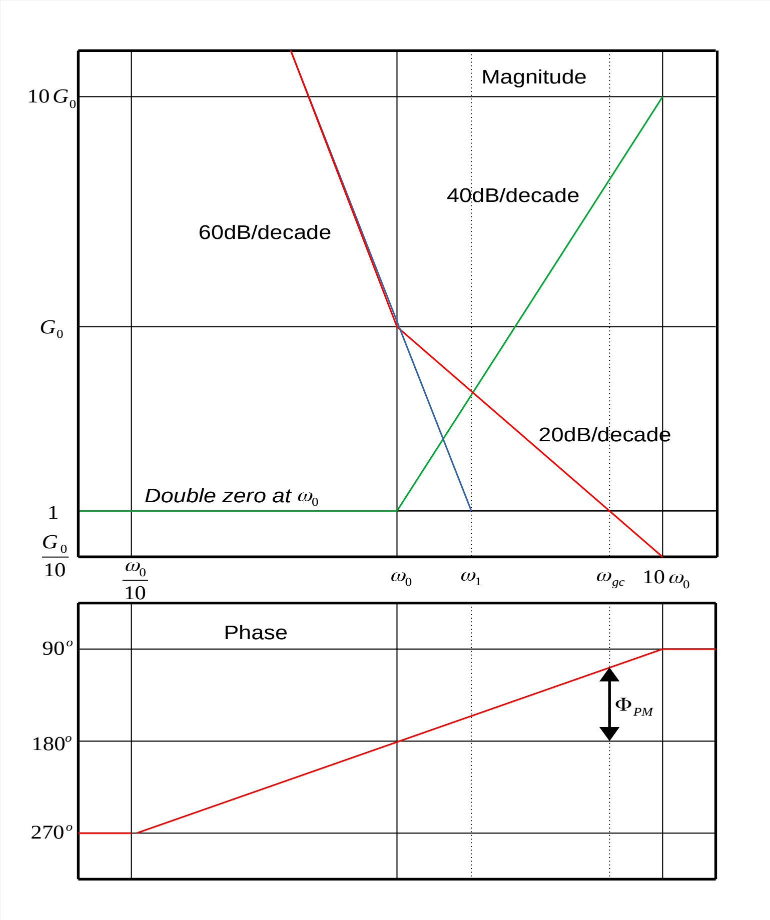
Type 3 filter has a more complicated general form, but according to Gardner (2005) the above form is sufficient for most cases.

Open loop transfer characteristics

Type 2 loop filter



Type 3 loop filter



PLLpy

Noise in PLL

One sided noise bandwidth

$$B_L = \int_0^{\infty} \left| H(f)^2 \right| df$$

Phase variance of the NCO

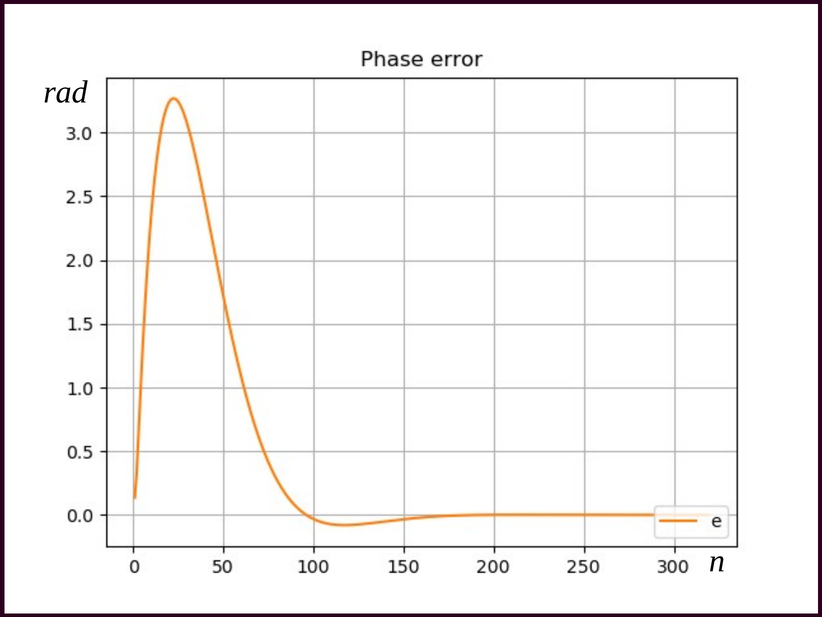
$$\sigma_{\theta no}^2 = \frac{W_0 B_L}{P_s}$$

W_0 : input noise power
P_s : input signal power

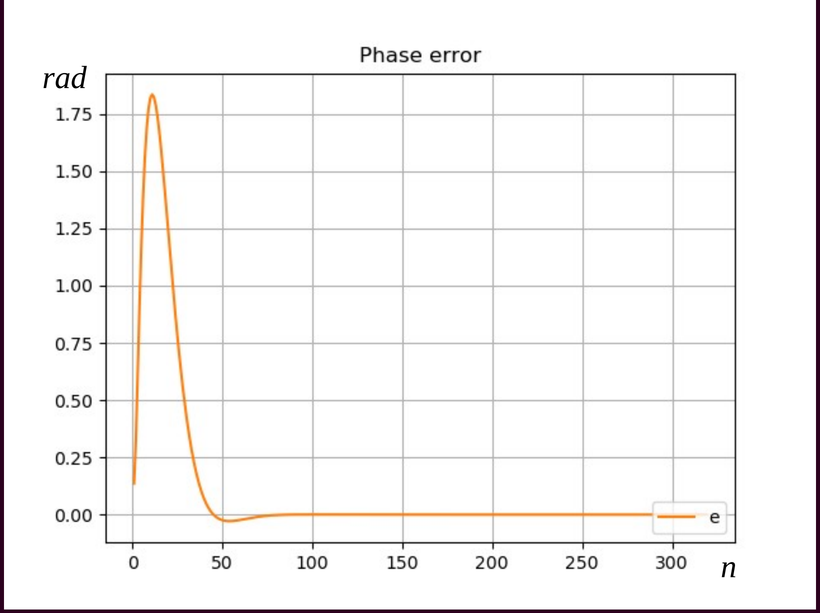
The phase noise of the NCO decreases with lowering the noise bandwith

PLLpy Loop bandwith and locking

Locking width BL=4Hz and df=8Hz
type2 loop filter



Locking width BL=8Hz and df=8Hz
type2 loop filter

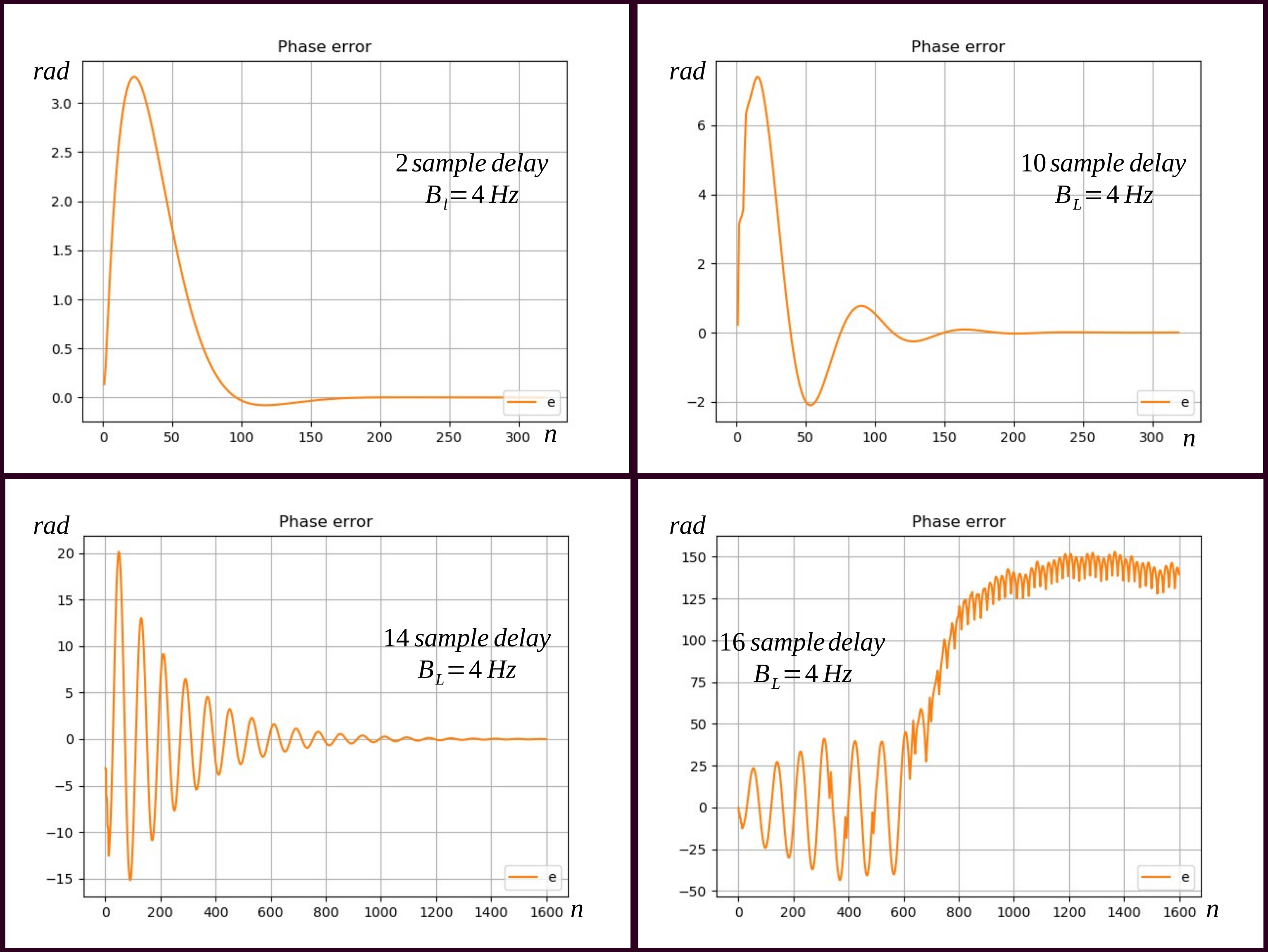


With increasing bandwidth
- The PLL locks faster
- The maximum phase error at given df is decreasing

PLLpy

Delay in the loop

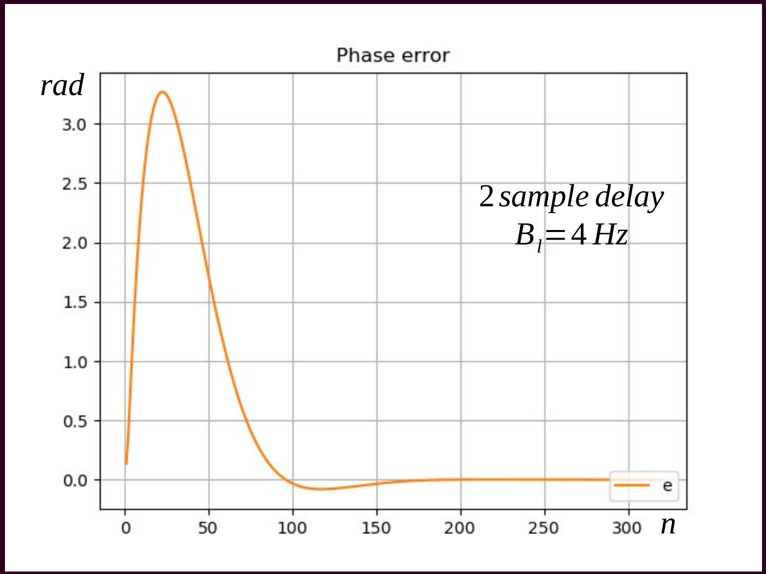
fs=200000Hz, Nfir=5000, decimation 1250, type2 loop filter

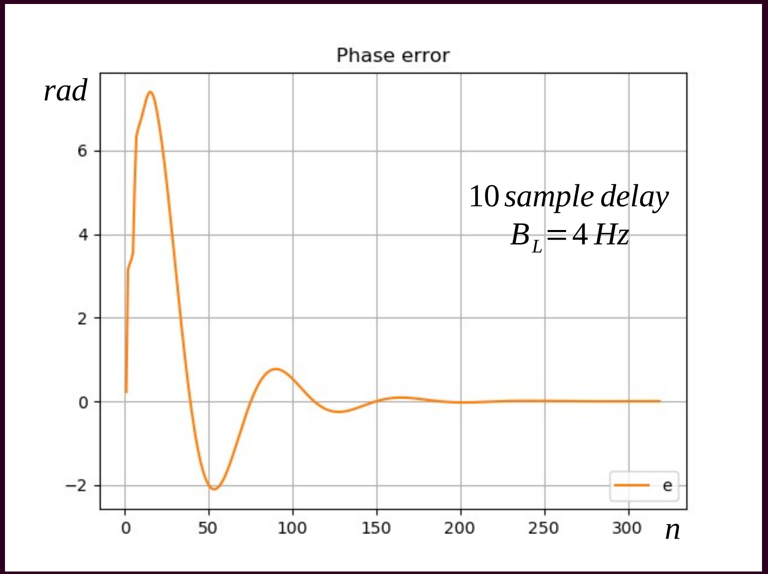


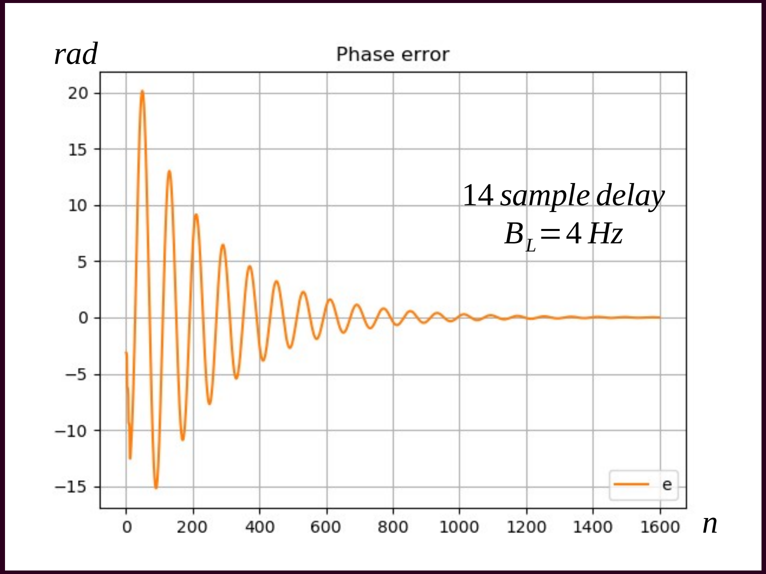
Increasing the delay the loop becomes unstable and finally will not lock

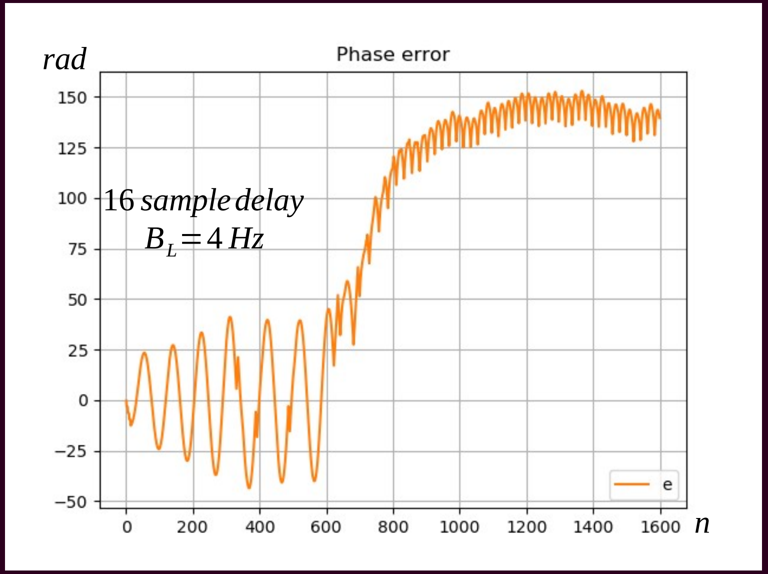
Delay in the loop

fs=200000Hz, Nfir=5000, decimation 1250, type2 loop filter







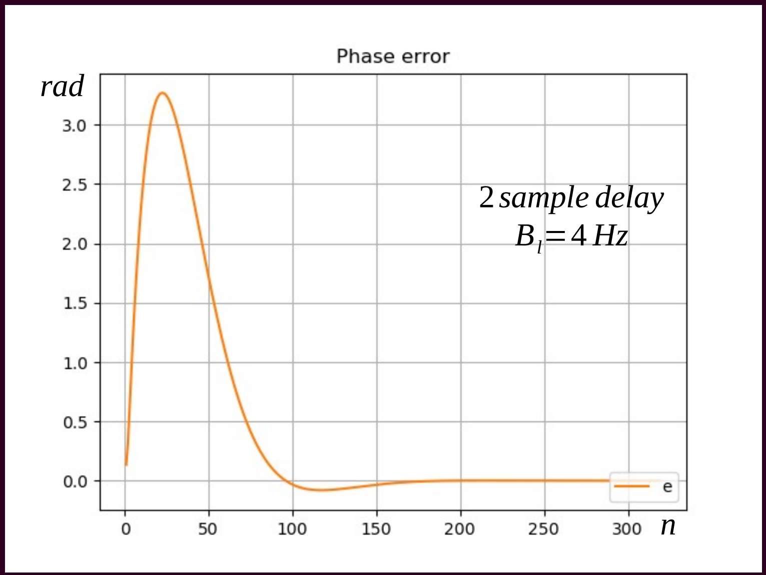


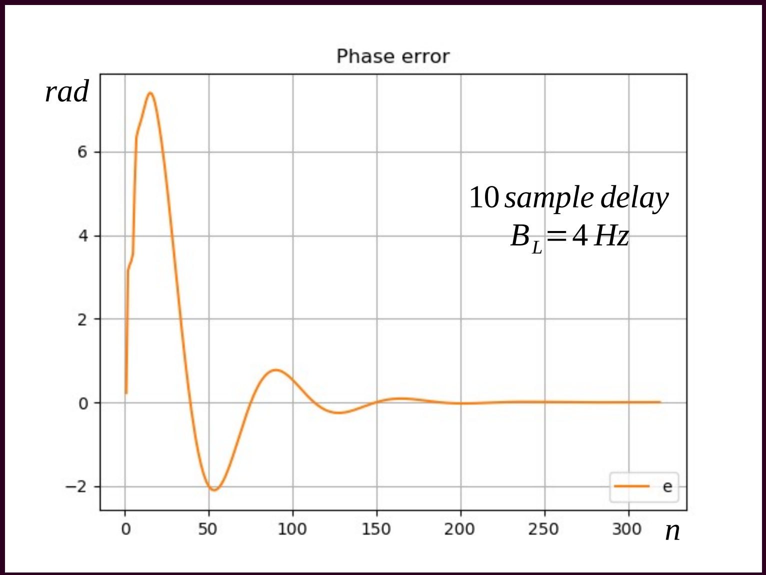
Increasing the delay the loop becomes unstable and finally will not lock

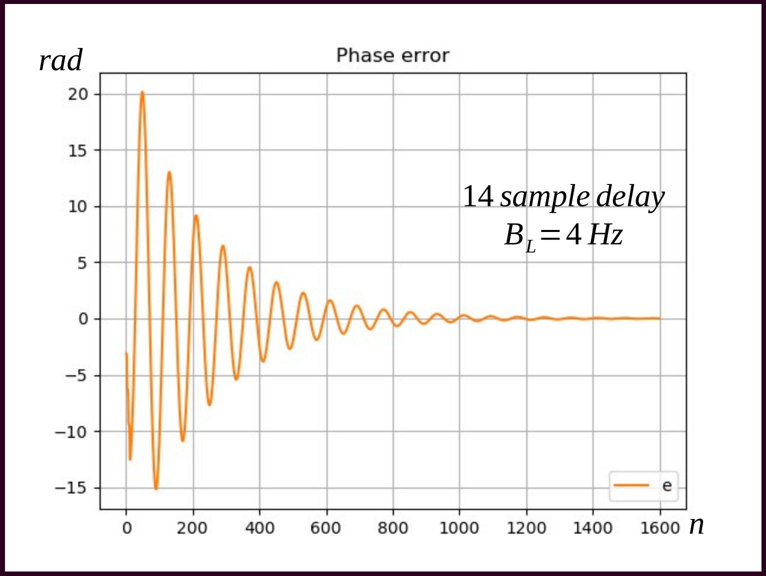
PLLpy

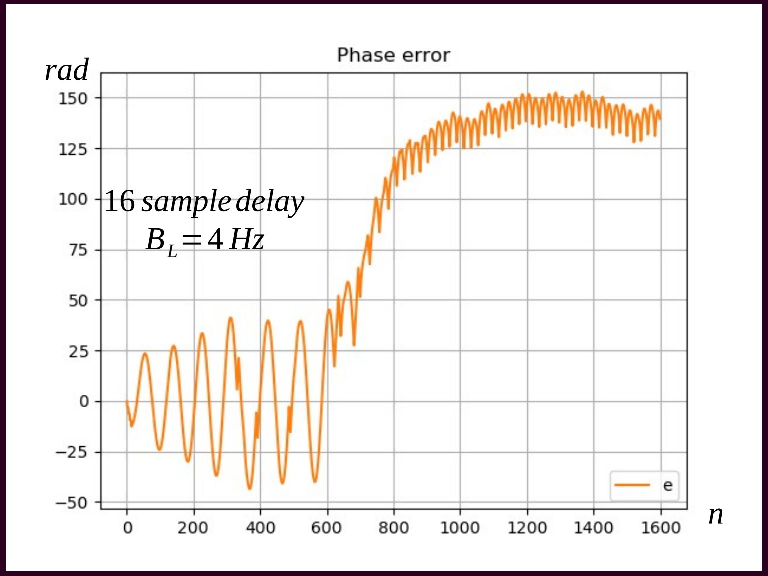
Delay in the loop

fs=200000Hz, Nfir=5000, decimation 1250, type2 loop filter





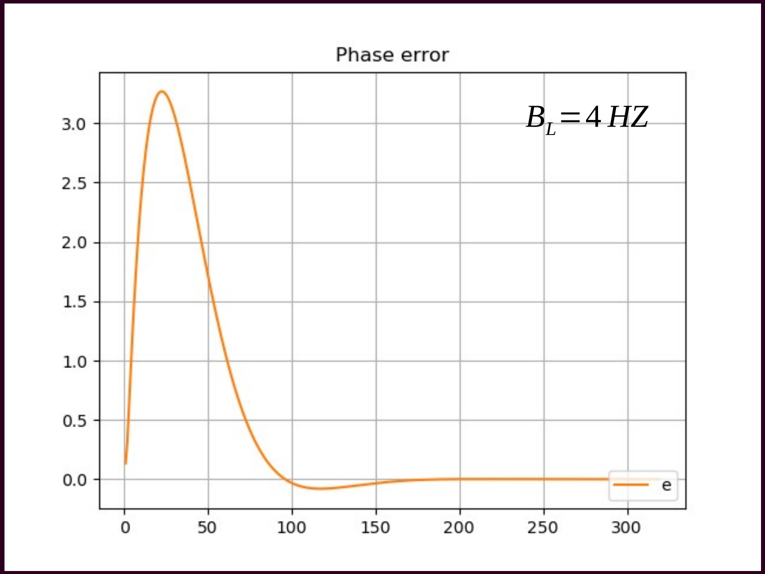


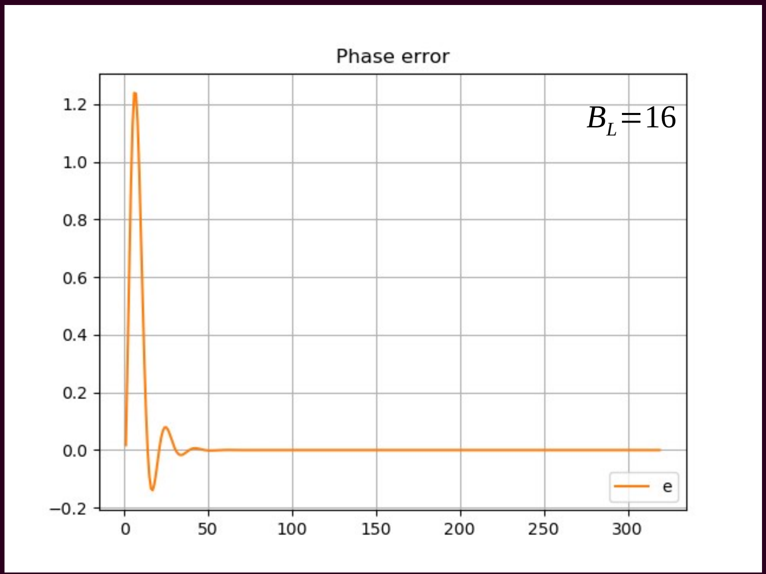


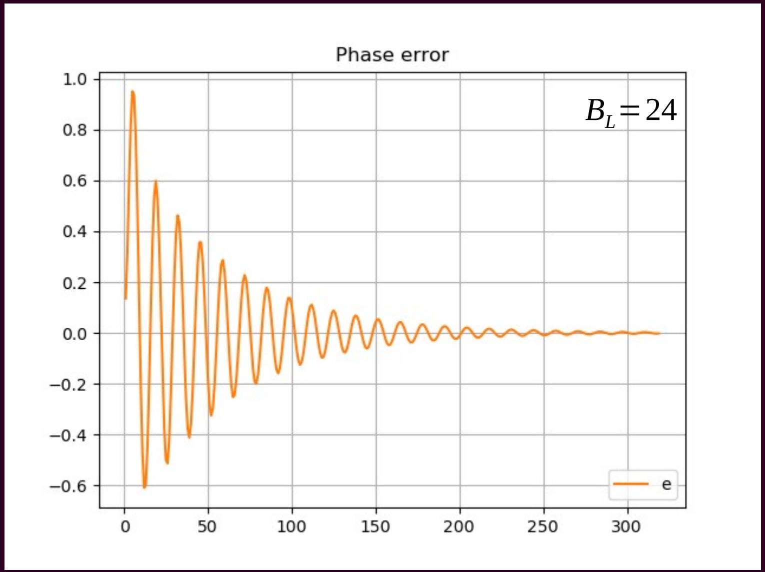
Increasing the delay the loop becomes unstable and finally will not lock

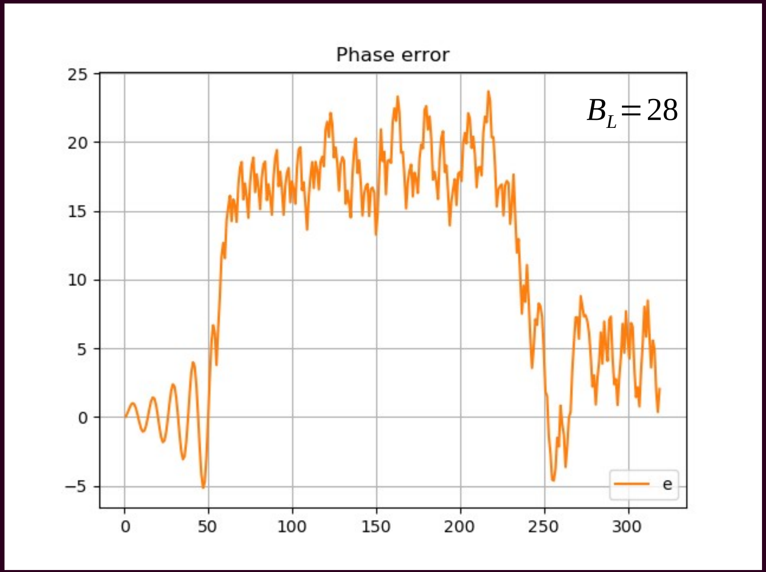
PLLpy Delay in the loop

Fs=200000, Nfir=5000, decimation factor 1250, type2 loop filter, 2 sample delay







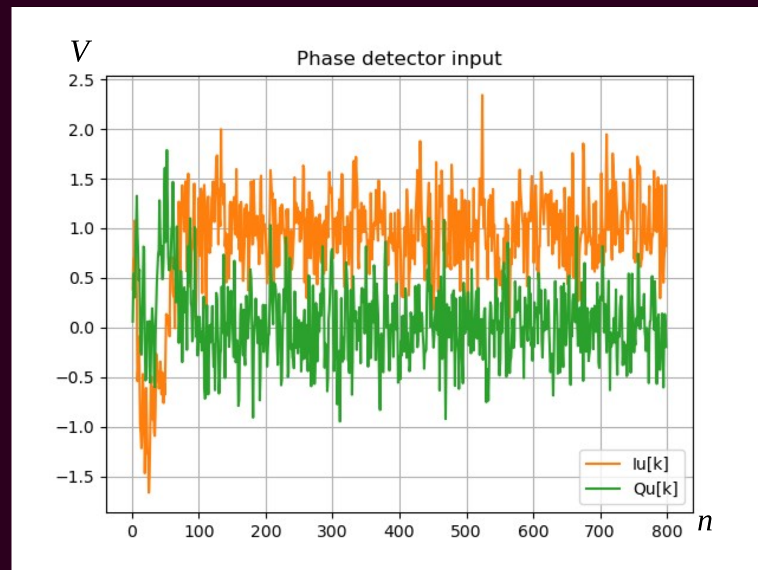


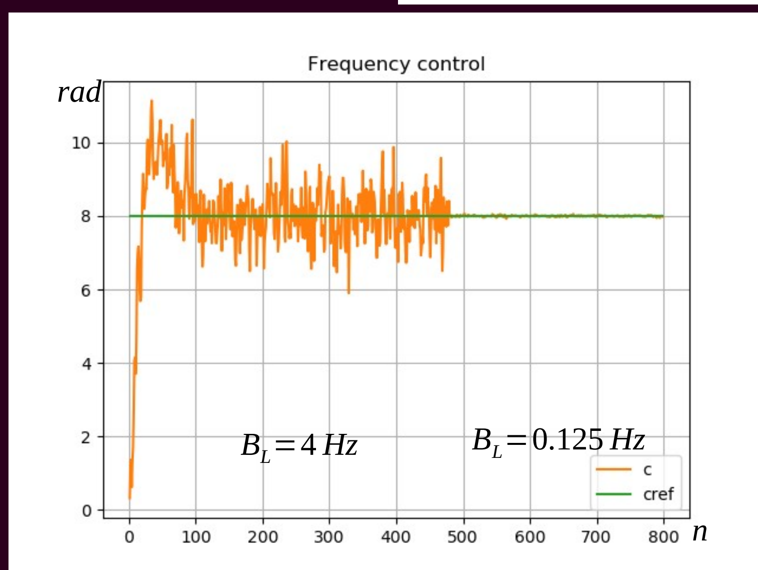
Increasing BL at a given delay the loop will be instable and finally does not locks

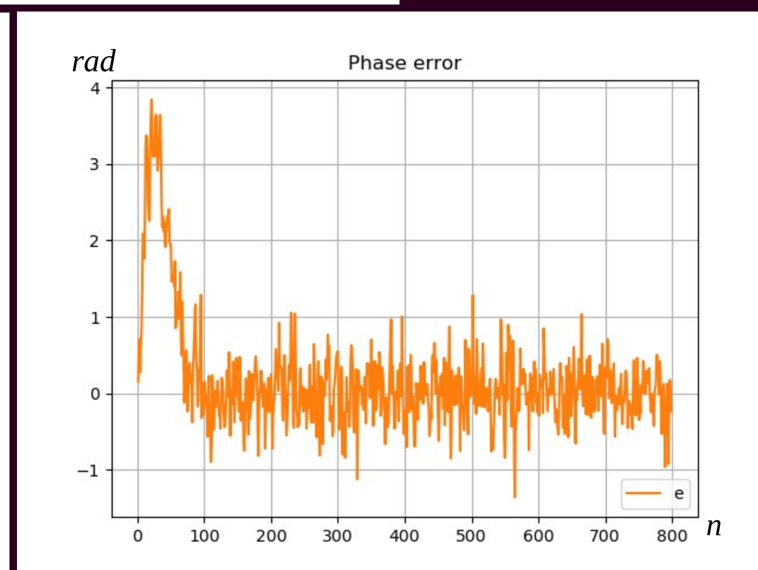
PLLpy

Noise in the loop

SNR=-27dB @ 200000KHz bandwidth, SNR=10dB @ phase detector input







Decreasing BL decreases the noise in the NCO control signal but not in the phase errror

PLLpy

Design formulas for loop filter

Designing from noise bandwidth and phase margin

$$\begin{gathered} Type\, 2\, PLL \\ \rho = \tan(\phi_{PM}) \\ K_P = 4 B_L \left(\frac{\rho}{1+\rho}\right) \\ \omega_0 = \frac{K_p}{\rho} \\ K_i = \omega_0 T_s \end{gathered}$$

$$\begin{gathered} Type\, 3\, PLL \\ \rho = \tan\left(\frac{\phi_{PM} + 90^o}{2}\right) \\ K_P = 4 B_L \left(\frac{2\rho - 1}{2\rho + 3}\right) \\ \omega_0 = \frac{K_p}{\rho} \\ K_i = \omega_0 T_s \end{gathered}$$

Higher noise bandwidth:

- faster locking
- larger NCO phase noise
- better tracking behavior

Higher phase margin:

- greater stability
- flatter settling curve

PLLpy

Tracking

Final value theoreme: $$\lim_{t \to \infty} y(t) = \lim_{s \to 0} sY(s)$$

Phase step: $$\theta_i = \frac{\Delta\theta}{s} \qquad \theta_p = \lim_{s \to 0} \frac{s\Delta\theta}{s+F(s)} = 0 \;\; if\; F(0) > 0$$

Frequency step: $$\theta_i = \frac{\Delta\omega}{s^2} \qquad \theta_v = \lim_{s \to 0} \frac{\Delta\omega}{s+F(s)} = 0 \;\; if\; F(0) = \infty$$

Frequency rump: $$\theta_i = \frac{\Lambda}{s^3} \qquad \theta_a = \lim_{s \to 0} \frac{\Lambda}{s(s+F(s))} = 0 \;\; if\; F(0) = \frac{Y(s)}{s^2}$$

The final value theoreme is used for the analyzis of the tracking behaviors

PLLpy

Tracking

Summary of the tracking behaviors

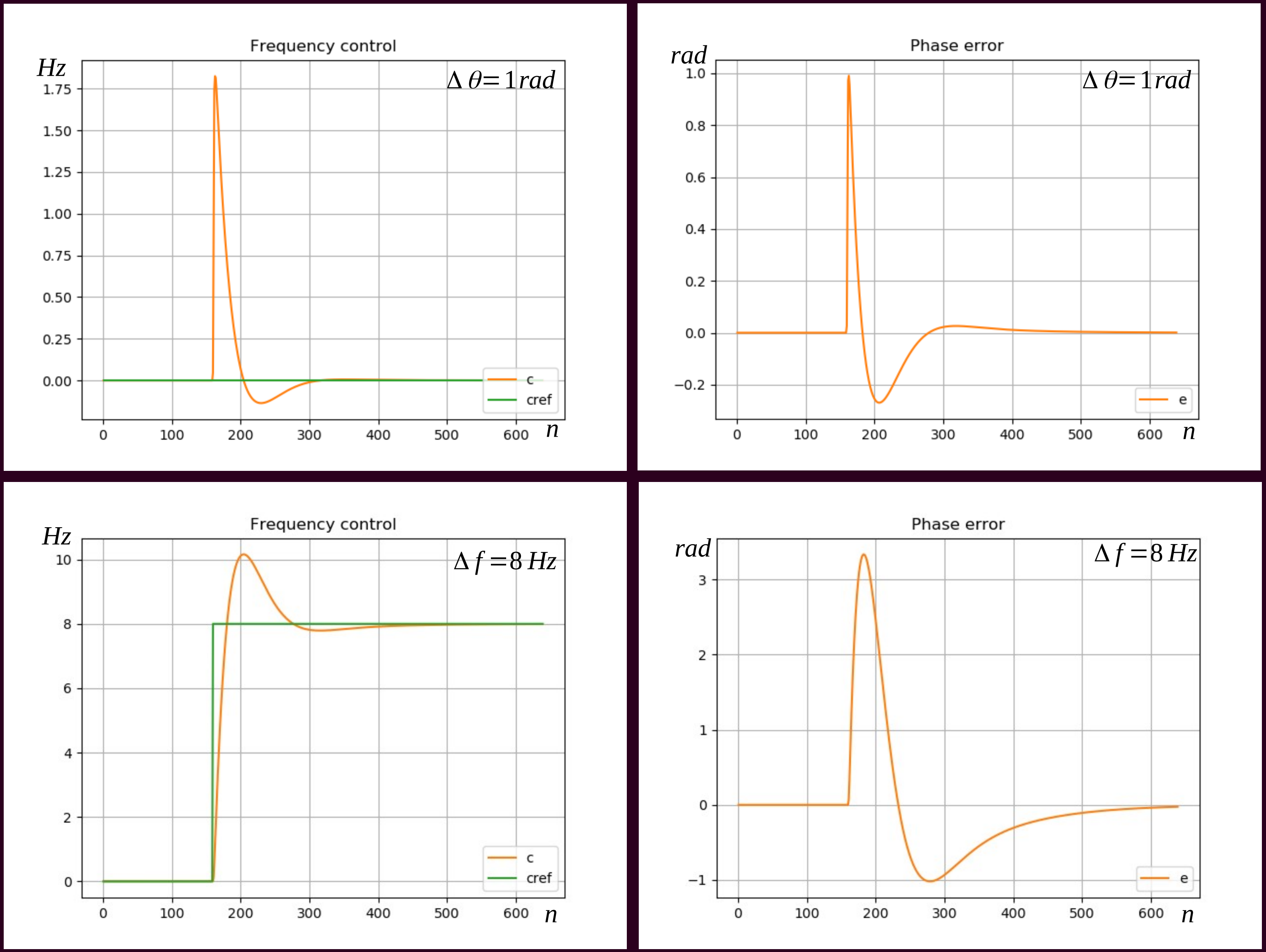
Input phase	Type 1 filter phase error	Type 2 filter phase error	Type 3 filter phase error
Phase step	0	0	0
Frequency step	Not zero	0	0
Frequency ramp	Not zero	Not zero	0
Frequency acceleration	Not zero	Not zero	Not zero

Phase error at frequencyacceleration for type 3 filter
increasing linearly with increasing acceleration
inverzely proportional with the third power of BL

PLLpy

Tracking phase and frequency step

fs=1e5, decimation=1250, type2 loop filter, BL=4

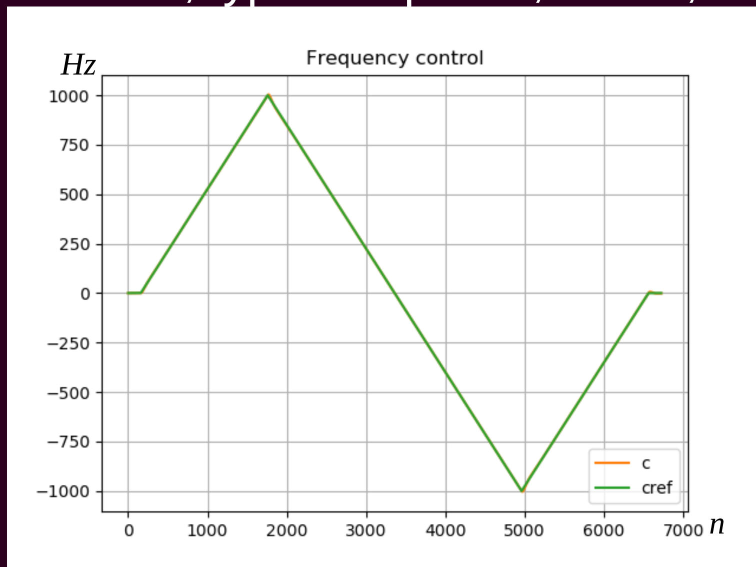


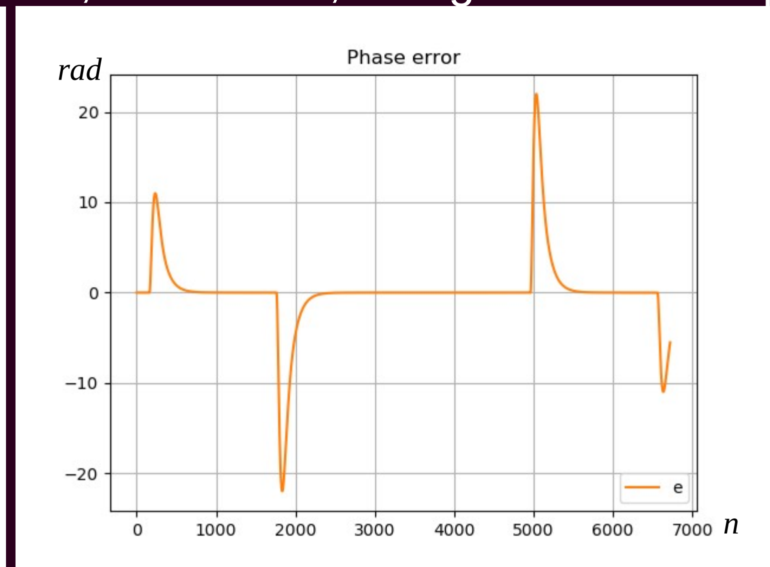
Type 2 loop filter tracks phase and frequency step with 0 phase error

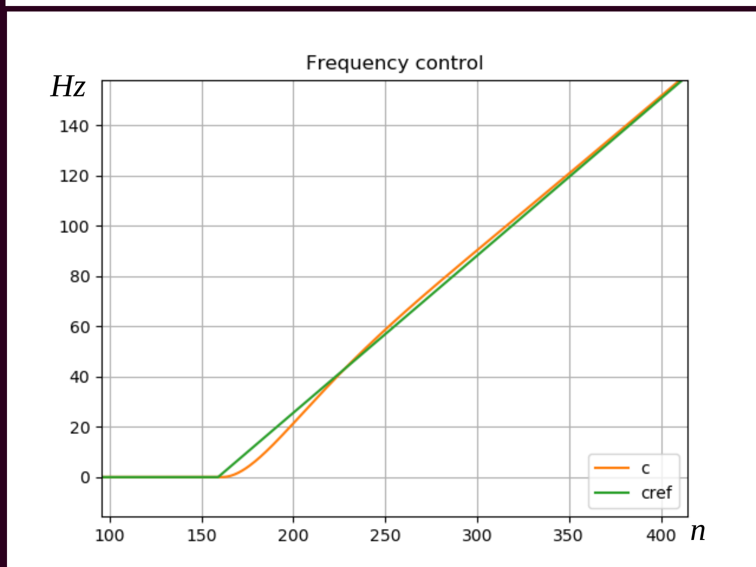
PLLpy

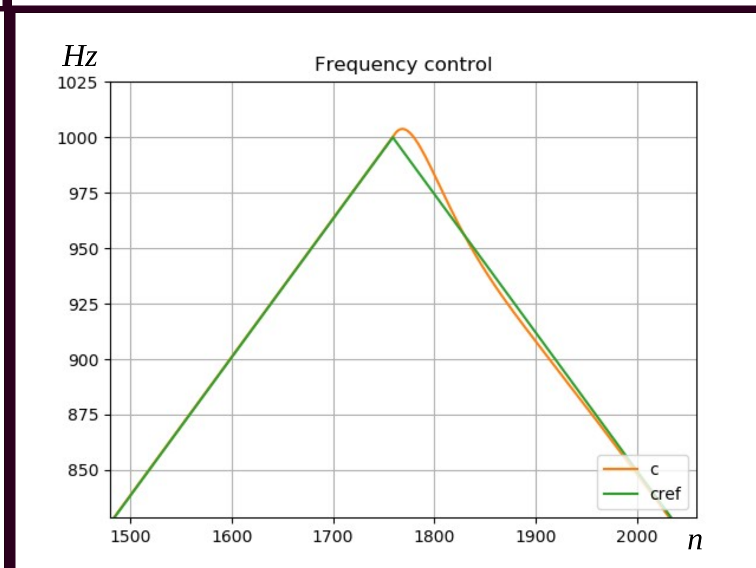
Tracking linear frequency change

BL=4Hz, type3 loop filter, fsw=0,025Hz, Asw=1000, triangle wave sweep







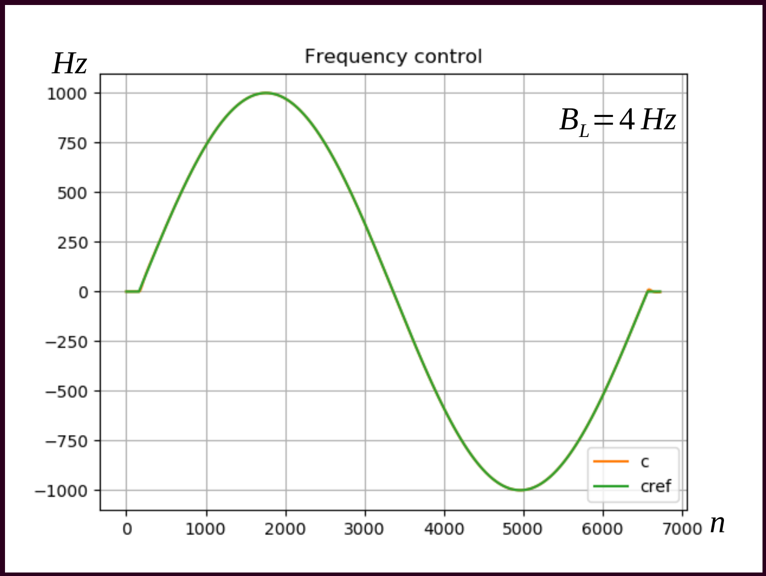


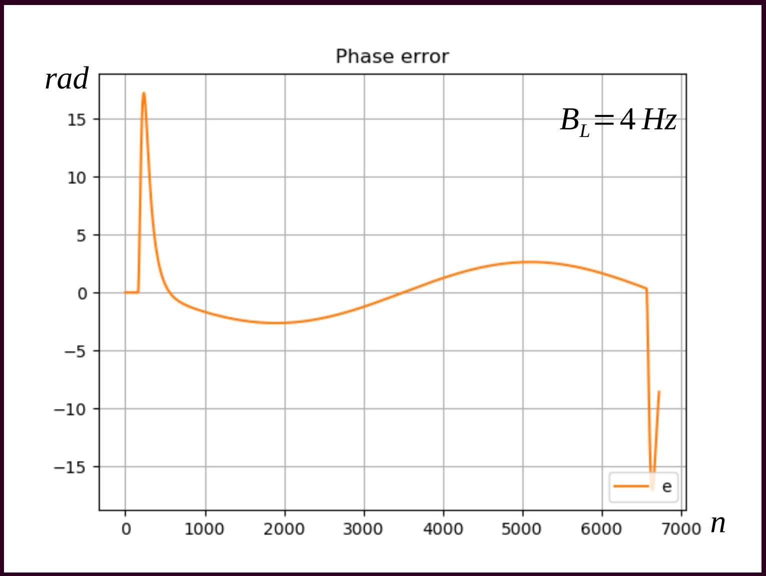
There are transients where the frequency slop sudenly changes

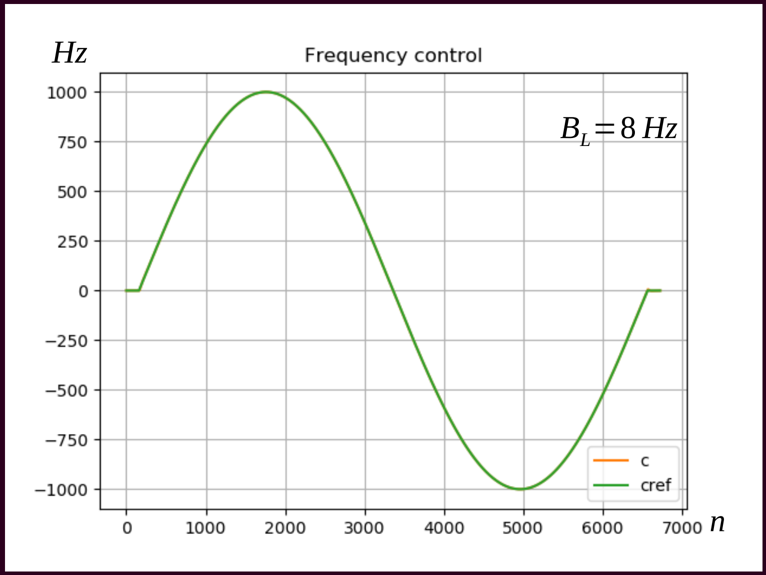
PLLpy

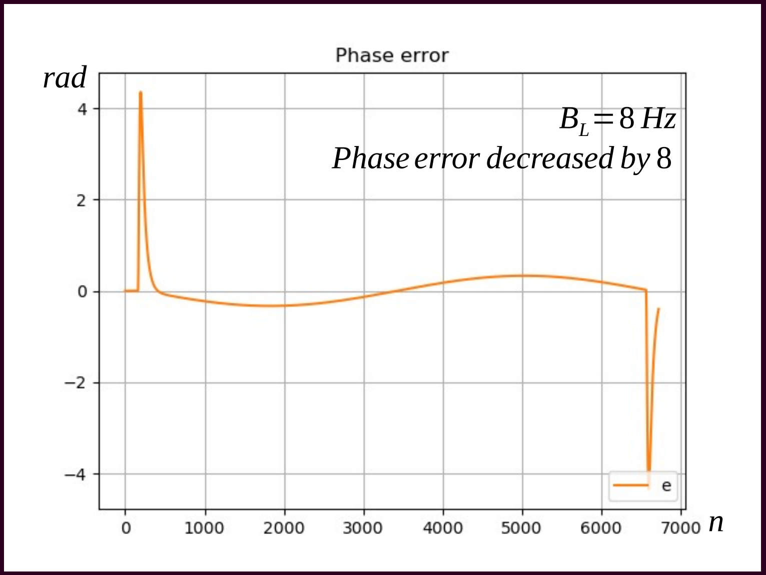
Tracking sine frequency change

BL=4Hz, BL=8,type3 loop filter, fsw=0.025Hz, Asw=1000, sine wave sweep









The phase error inversely proportional with the 3rd power of BL

PLLpy Further reading

- Farhang-Boroujeny, B. (2010), Signal Processing Techniques for Software Radios, 2nd edition
- Gardner, F. M. (2005), PhaseLock Techniques, 3rd edition, John Wiley & Sons
- Staszewski, R. B. and all (2011), Dynamic Bandwidth Adjustment of an RF All-Digital PLL, IEEE Xplore, July 2011.
- Hurd, W. J. (1970), Digital Transition Tracking Symbol Synchronizer for LOW SNR Coded Systems, IEE Transactions On Communication Technology, Vol. COM-18, No. 2, April 1970.
- Gardner, F. M. (1986), A BPSK/QPSK Timing Error Detector for Sampled Receivers, IEEE Transactions on Communications, vol. 34, no.5, pp. 423-429, May 1986.
- Prouzet A. H. (1972), Characteristic od phase detectors in presence of noise, International Telemetring Conference Proceeding, 1972

PLLpy

Algorithms in pllpy

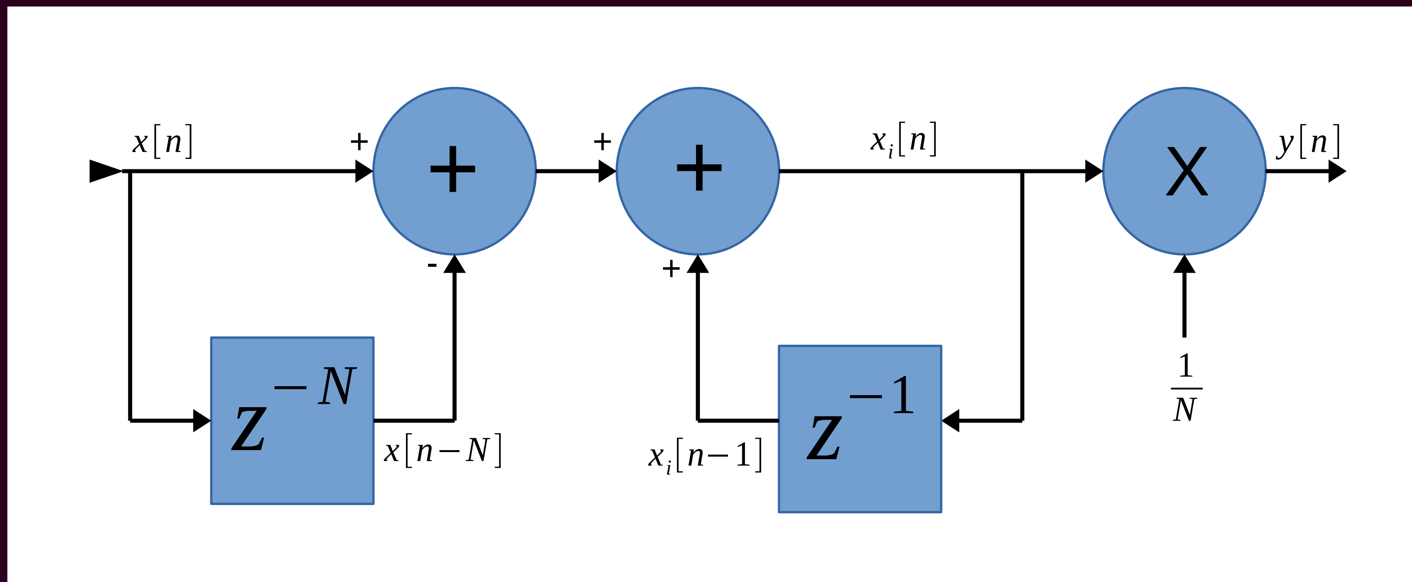
Simple algorithms
- Complex NCO
- Square Wave NCO
- Rectangular Pulse PAM Generator
- BPSK Modulator
- Complex Noise Generator
- Complex Channel
- Complex Moving Average Decimator
- Complex FIR Decimator
- Complex Variable Moving Average Filter
- Moving Average Filter
- Complex Moving Average Filter
- Complex Variable Moving Average Filter
- Gated Integrator
- Phase Detector
- Edge Phase Detector
- FFT Frequency Detector
- Type 2 Loop Filter
- Type 3 Loop Filter

Loops
- PLL or Costas Loop
- Frequency Locked Loop (FLL)
- Bit Recovery Loop

Loop tests
- PLL or Costas Loop Test
- Frequency Locked Loop Test
- Bit Recovery Loop Test

The simple algorithms and loops
- work in sample by sample bases
- have uniform external interface

PLLpy Moving Average Filter



N is the length of the filter

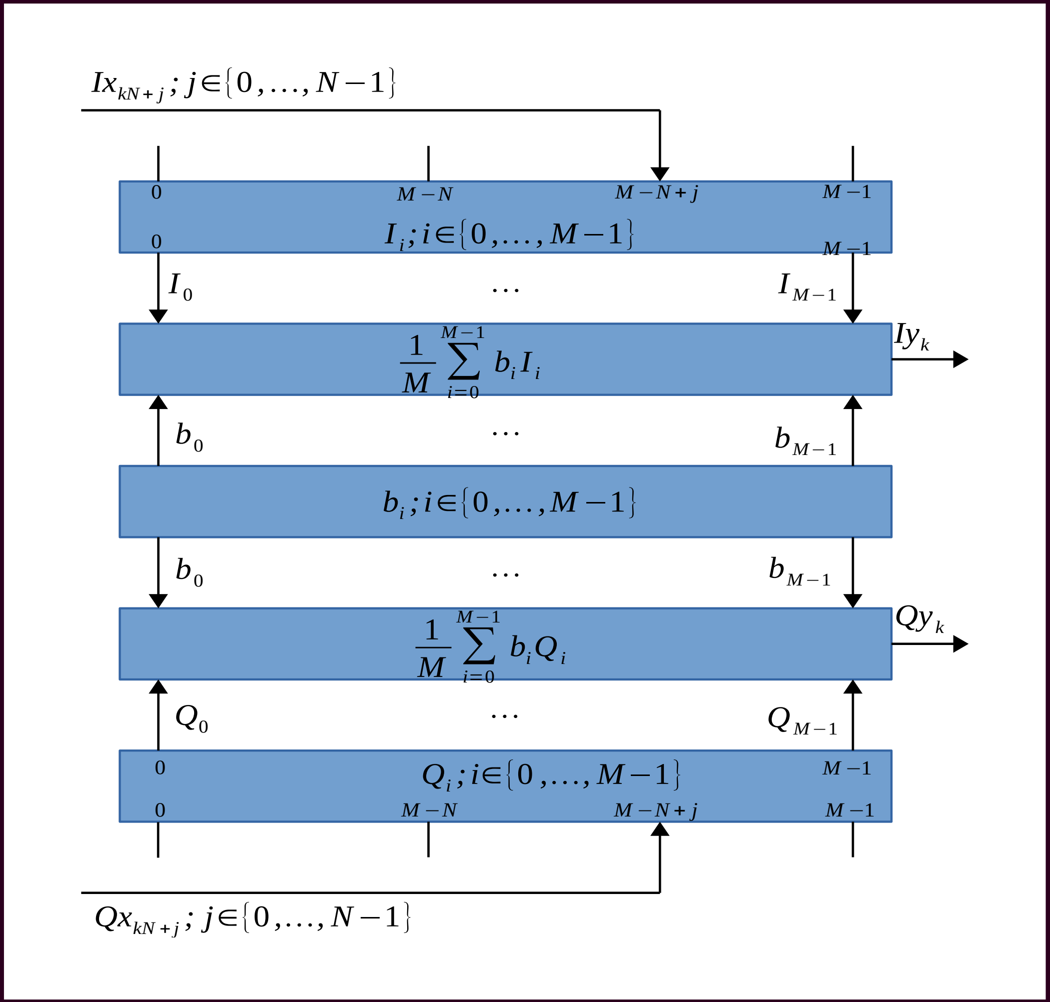
$$x_k = 0 \quad -N < k \leqslant -1$$

$$y_n = \frac{1}{N} \sum_{i=0}^{N-1} x_{n-i}$$

$$f_{cutoff} \quad f_s \frac{0.443}{N}$$

The transfer characteristics in the frequency domain is a sync function

FIR Decimator



M is the length of the filter
N is the decimation factor

$$Iy_k = \frac{1}{M}\sum_{i=0}^{M-1} b_i Ix_{(k+1)N-1-i}$$

$$Qy_k = \frac{1}{M}\sum_{i=0}^{M-1} b_i Qx_{(k+1)N-1-i}$$

In pllpy it is designed

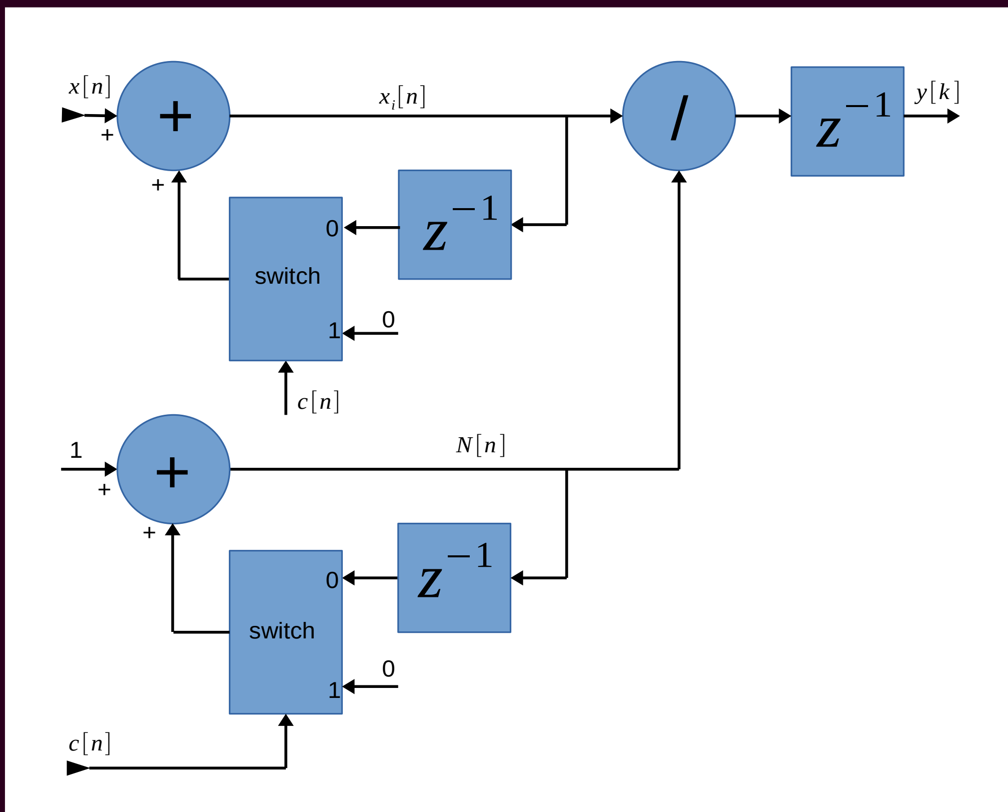
from brick wall frequency response and

Blckman – Harris time domain windowing

It has M/(2*N) decimated sample group delay

PLLpy

Gated integrator



The first c[n]=1
- starts accumlating and counting
  - the incomming samples

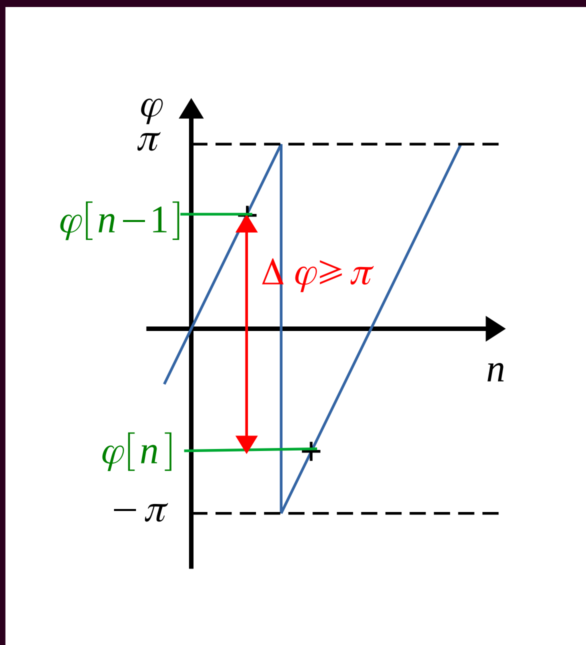
The following c[n]=1
- Stops the conting and
  - accumlating process
- Finalizes the output data
- Starts the next processing cycle

This is rather a moving average filter whose length is determined by the gating signal

PLLpy

Phase detector

Works in unwrapping and Principal value mode

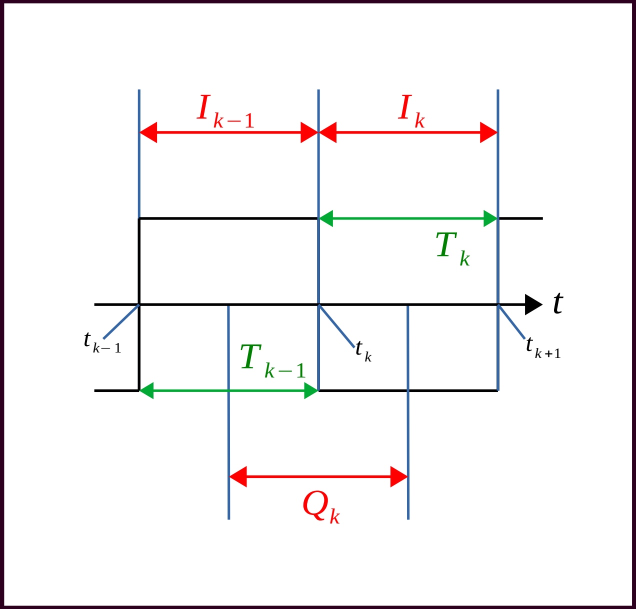


$$x: input\ complex\ number$$
$$z = x \quad if\ normal\ type$$
$$z = x^2 \quad if\ BPSK\ type$$
$$z = x^4 \quad if\ QPSK\ type$$
$$\phi = arc(z) \quad this\ is\ Principal\ Value$$
$$\phi_{out} = \phi_{unwrap} \quad if\ normal\ mode$$
$$\phi_{out} = 0.5\,\phi_{unwrap} \quad if\ normal\ mode$$
$$\phi_{out} = 0.25(\phi_{unwrap} - \pi) \quad if\ normal\ mode$$

$$(\phi_n - \phi_{n-1} < -\pi) \wedge (\phi_{n-1} > \pi/2) \wedge (\phi_n < -\pi/2) \Rightarrow \Phi_{acc} = \Phi_{acc} + 2\pi$$
$$(\phi_n - \phi_{n-1} > \pi) \wedge (\phi_{n-1} < -\pi/2) \wedge (\phi_n > \pi/2) \Rightarrow \Phi_{acc} = \Phi_{acc} - 2\pi$$
$$\phi_{unwrap} = \phi_n + \Phi_{acc}$$

In case of low input SNR unwrapping should not be used, have to operate in Principal Value mode

PLLpy Edge Phase Detector



T_k *is the kth bit length*

$$t_k = \sum_{i=0}^{K-1} T_k$$

$$I_k = \frac{1}{N_k} \sum x_n \quad t_k \leqslant nT_s < t_{k+1}$$

$$Q_k = \frac{1}{N_k} \sum x_n \quad t_k - \frac{T_{k-1}}{2} \leqslant nT_s < t_{k+1} - \frac{T_k}{2}$$

$$e_k = -sign(I_{k-1}) Q_k \quad if\ sign(I_{k-1}) \neq sign(I_k)$$

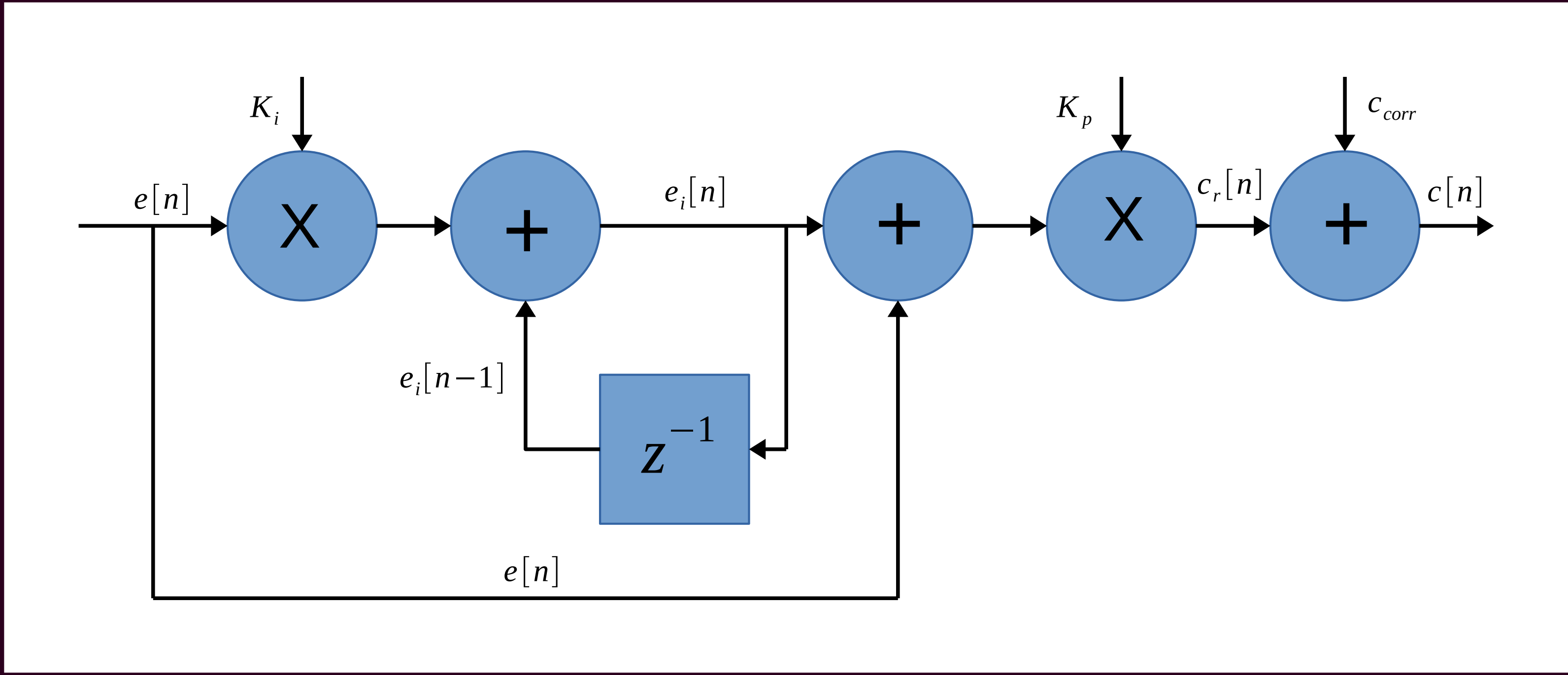
$$e_k = 0 \quad otherwise$$

The detector input is: I_k, Q_k

The integration should be done outside the detector

Equivalent to Gardner (1986) solution

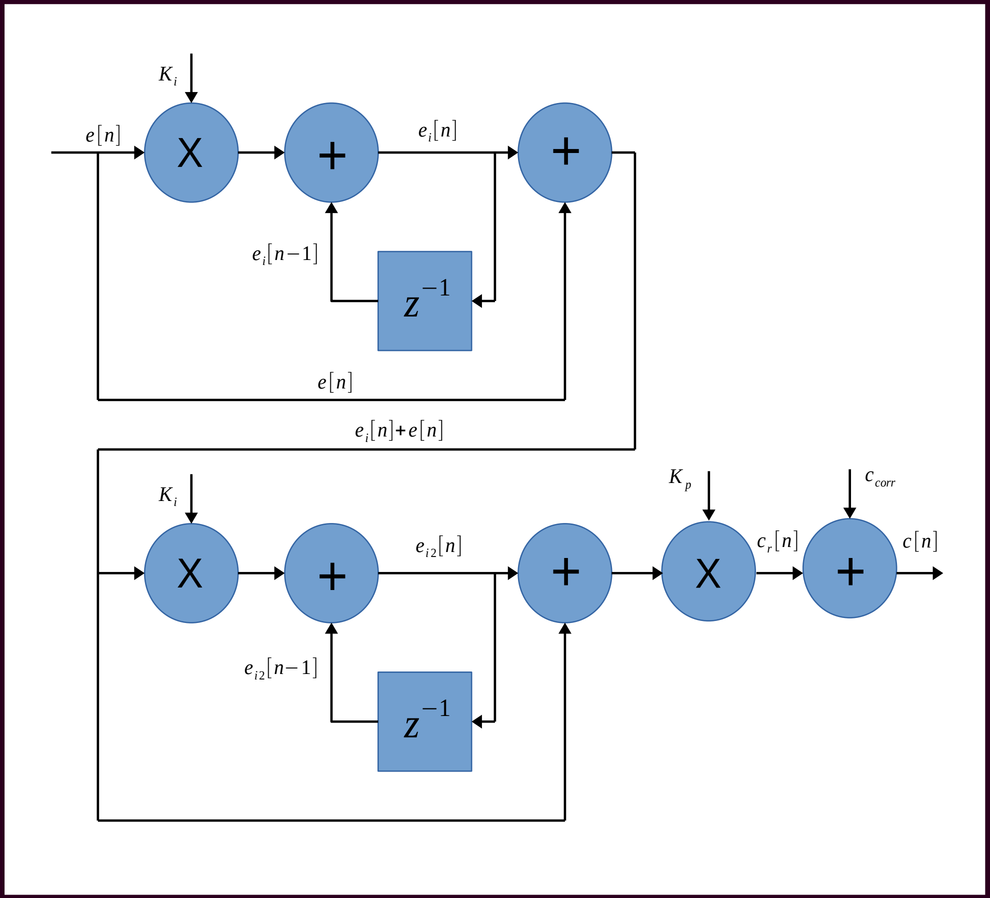
PLLpy Type 2 Loop Filter



$$F(z) = K_p\left(1 + \frac{K_i}{1 - z^{-1}}\right)$$
$$e_i[n] = e_i[n-1] + K_i e[n]$$
$$c[n] = K_p(e[n] + e_i[n]) + c_{corr}$$

The real integration gain in the usual topology is Kp*Ki

PLLpy Type 3 Loop Filter

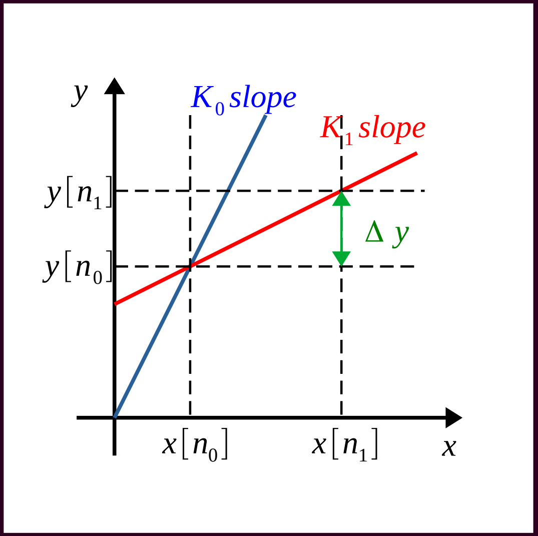


$$F(z)=K_p\left(1+\frac{K_i}{1-z^{-1}}\right)$$

$$e_i[n]=e_i[n-1]+K_i e[n]$$

$$c[n]=K_p(e[n]+e_i[n])+c_{corr}$$

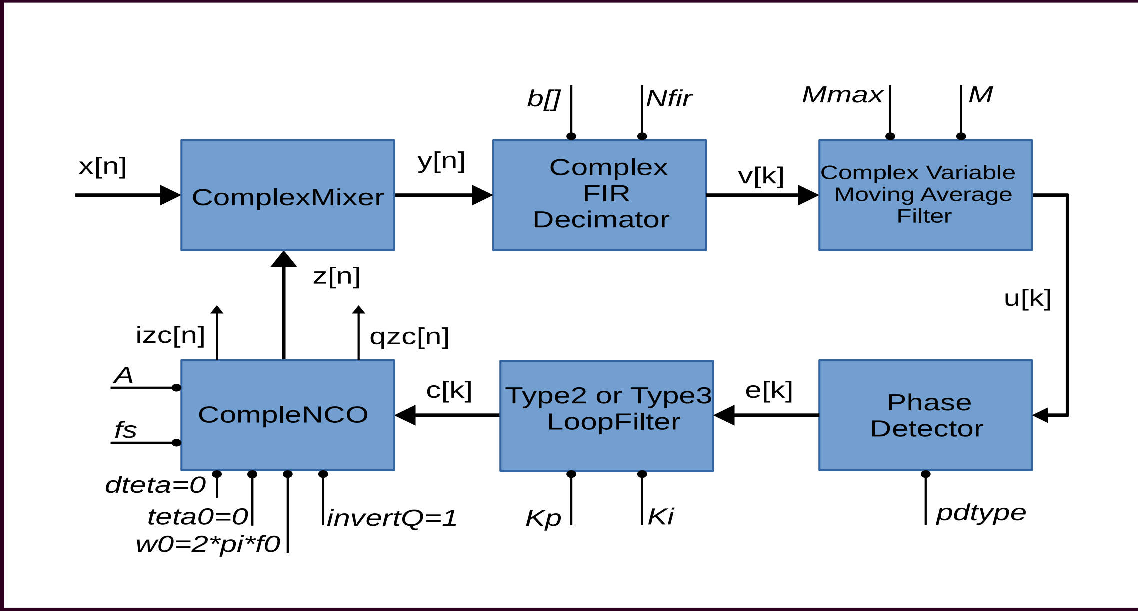
PLLpy Loop Filter Kp,Ki change



- Ki could be changed freely
- For Kp change special solution needed
- The slope should be changed at the bias point
- The average y value should specify the bias point

Basic idea came from Staszevwki (2011), the implementation has been modified for working with consecutive bandwidth changes and for preventing tranzients in the presence of noise.

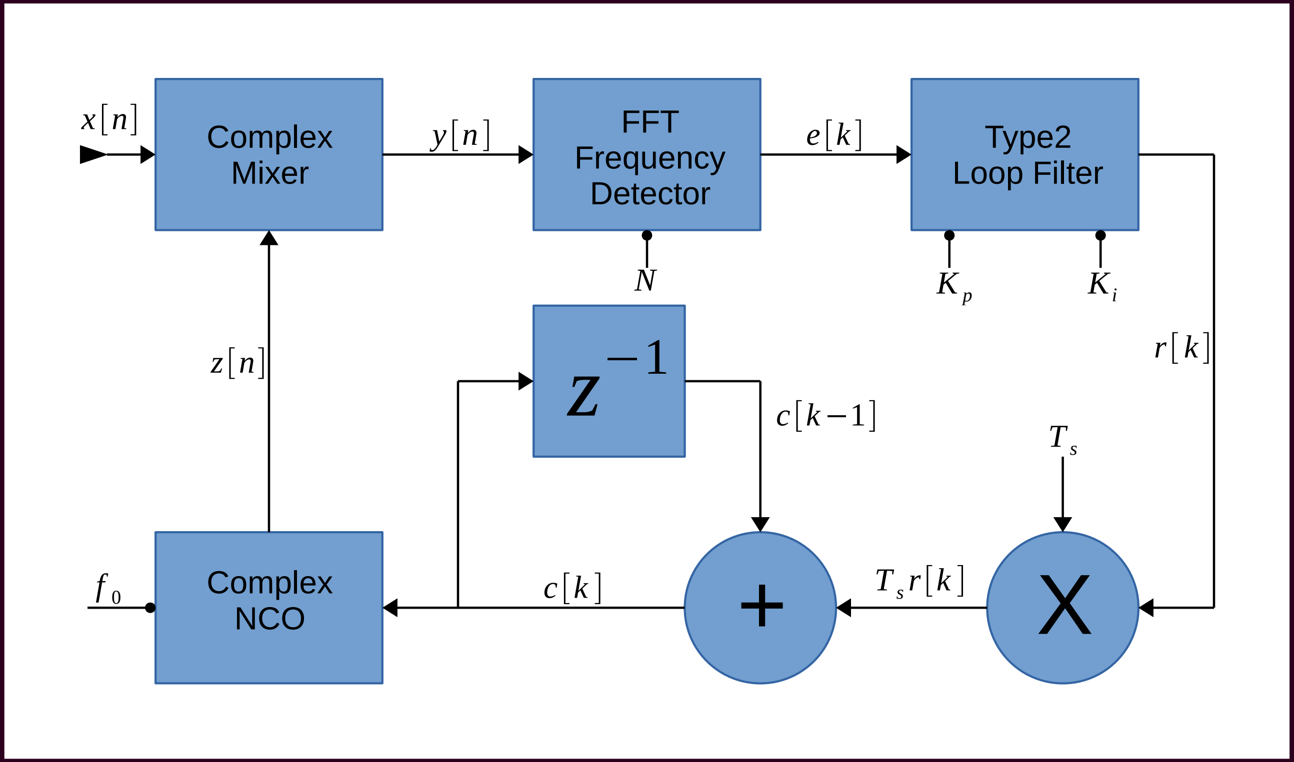
PLLpy PLL or Costas Loop



Pdtype determine if this is a
- carrier loop
- Costas loop for BPSK
- Costas loop for QPSK

The topology on the drawing is different from the usual Costas loop topology

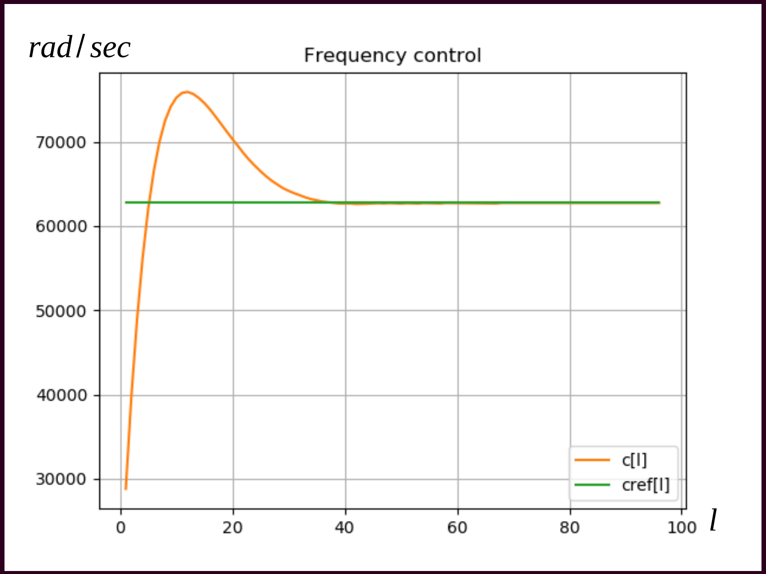
PLLpy Frequency Locked Loop

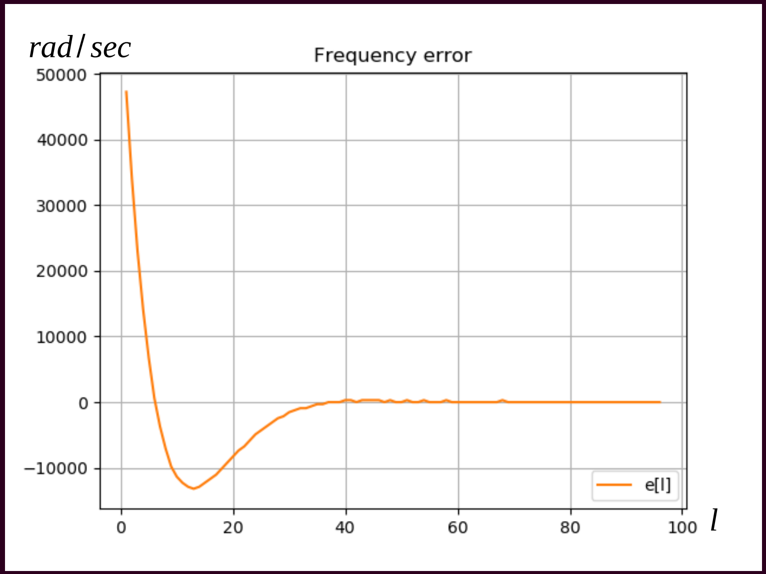


We need an integrator following the loop filter because the error is frequency error and not a phase error.

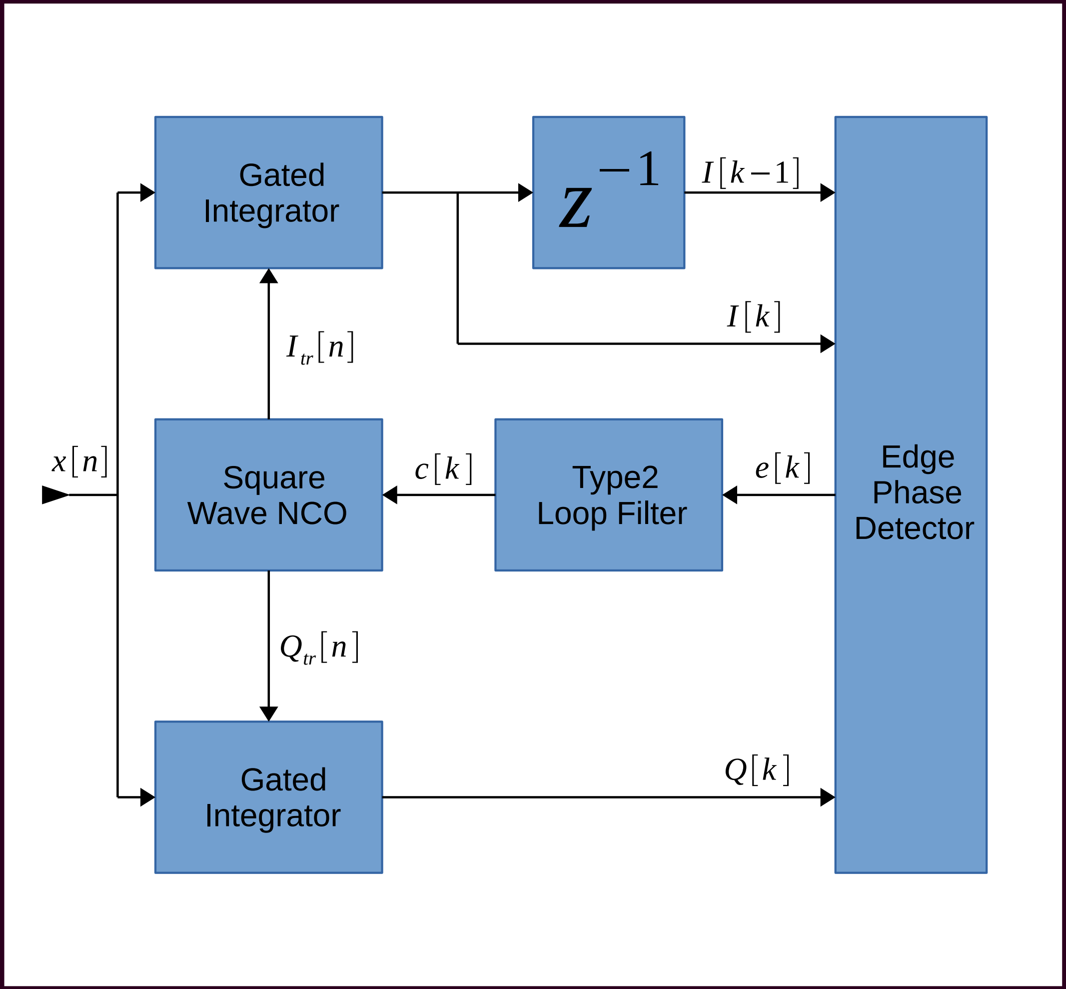
FLL results

df=1e4, BL=4Hz, Nfft=4096





PLLpy Bit Recovery Loop

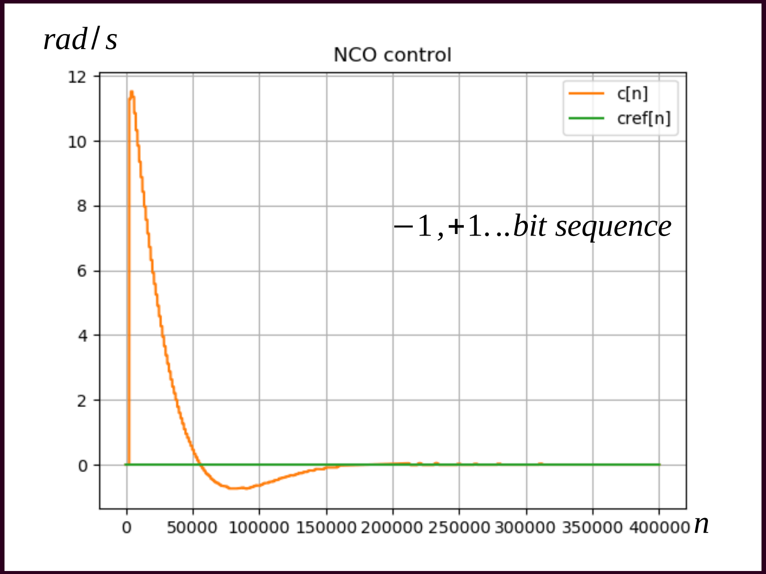


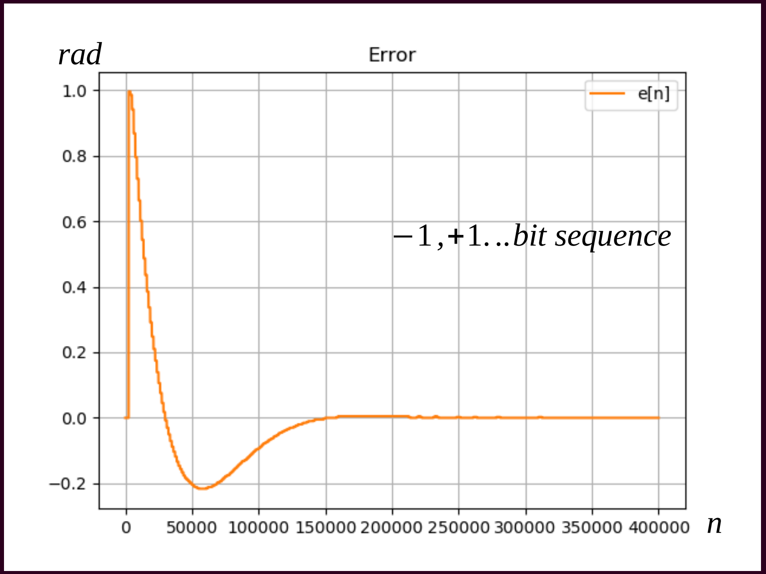
The gated integrators are matched filters and samplers.
Equivalent to Gardner (1986). Based on Hurd (1970). Used in many ESA transponders.

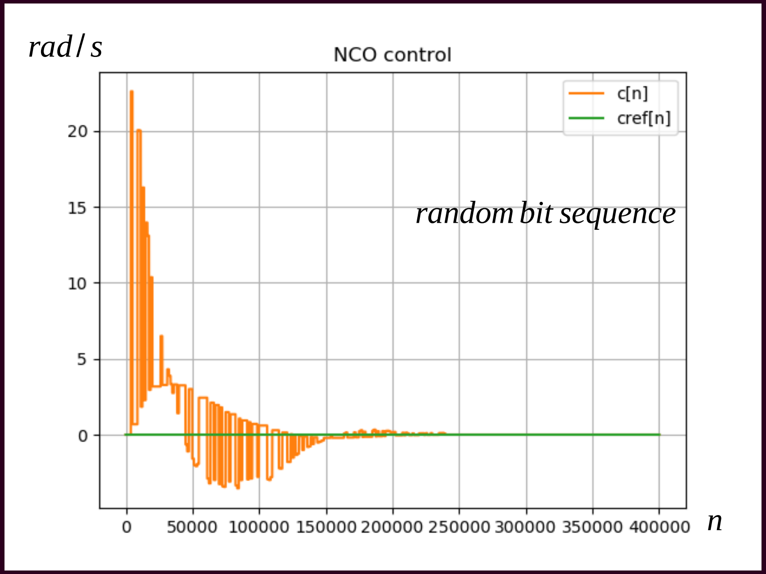
PLLpy

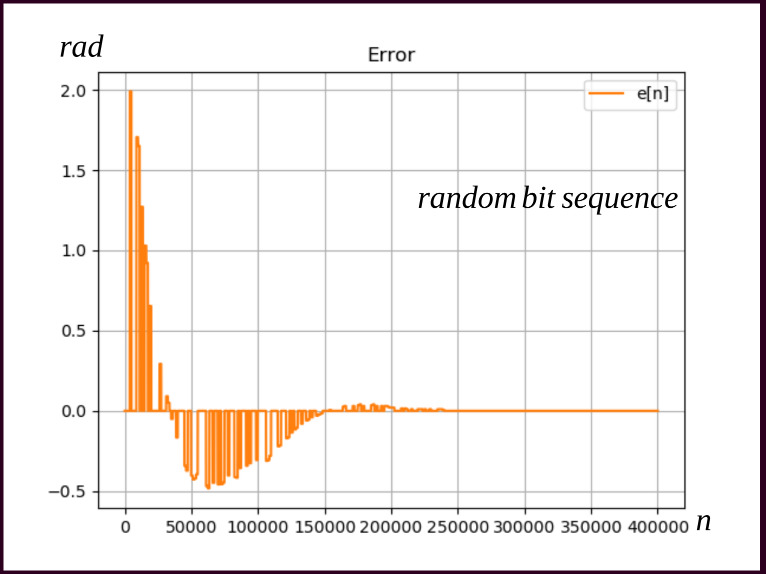
Bit recovery results

dteta=1rad, BL=4Hz, fbit=160Hz







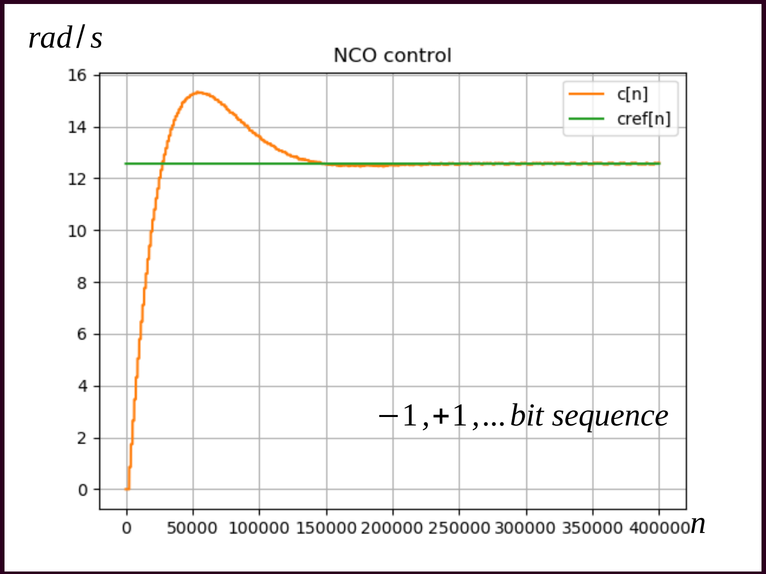


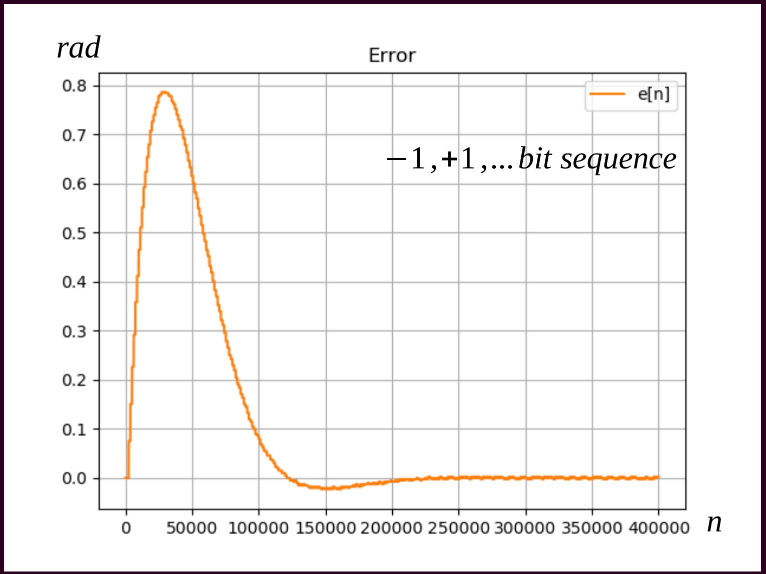
In case of random bit sequence the error from the detector is multiplied by 2 to compensate the absent of edge in many measurement.

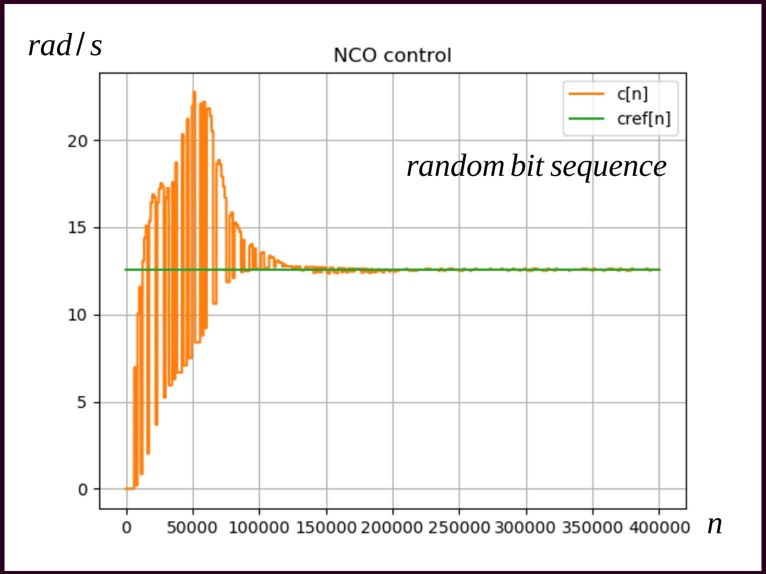
PLLpy

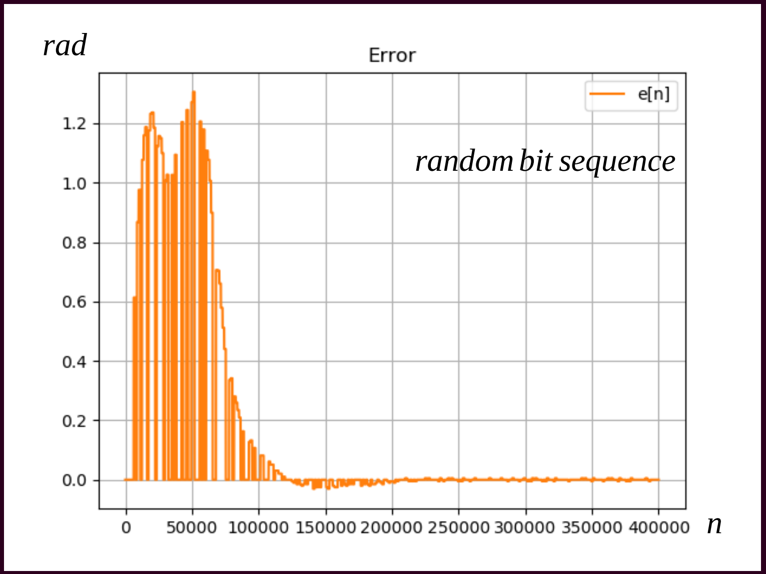
Bit recovery results

df=2Hz, BL=4Hz, fbit=160Hz









PLLpy Getting started with pllpy

I recommend to use ubuntu 16.4 linux system
First you should install the pre-requisites:

```
> sudo apt-get install git python3 python3-numpy
> sudo apt-get install python3-scipy python3-matplotlib
```

Next you shoul get pllpy from github
Create a work directory if you already do not have one and go into this directory

```
> mkdir work
> cd work
```

Clone the pllpy from github

```
> git clone https://github.com/ha5ft/pllpy
```

Go the work/pplpy directory an start python3
Now you are ready to try one of the test.

PLLpy

Run Your first test

Go to the pllpy directory where you cloned the software from github
Start the python3 consol

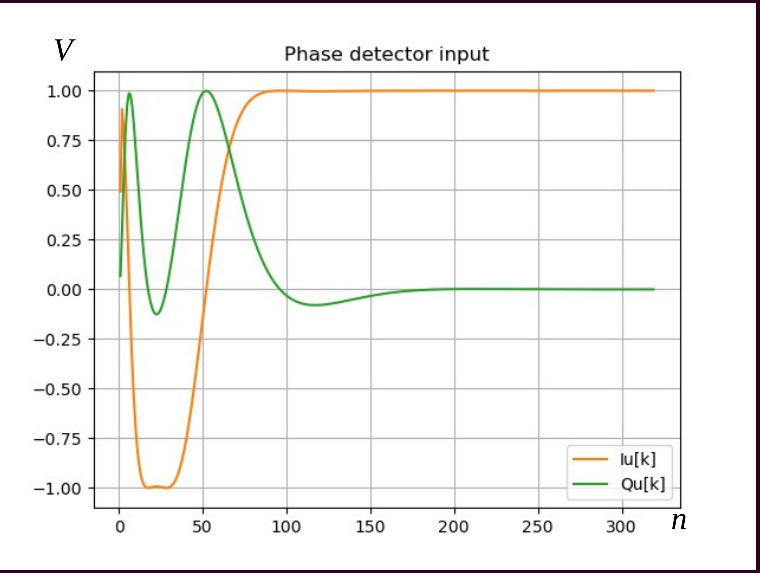
```
> python3
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

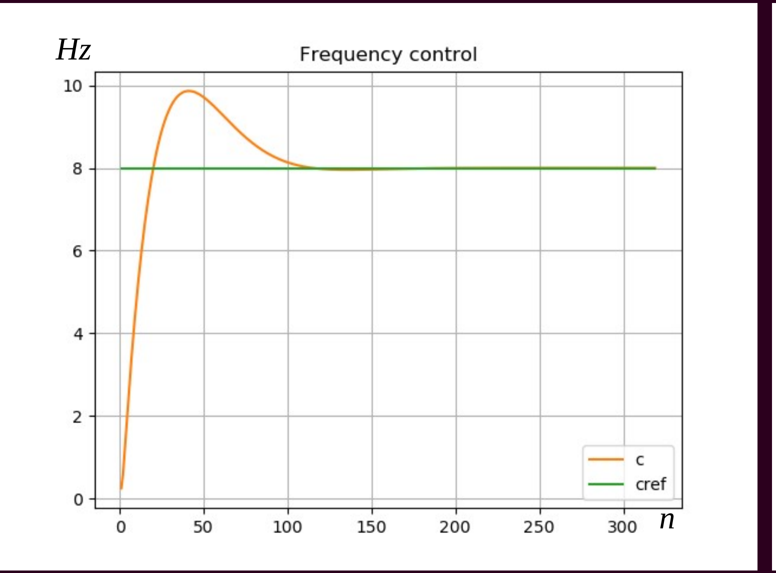
Now you are ready to try on of the test.

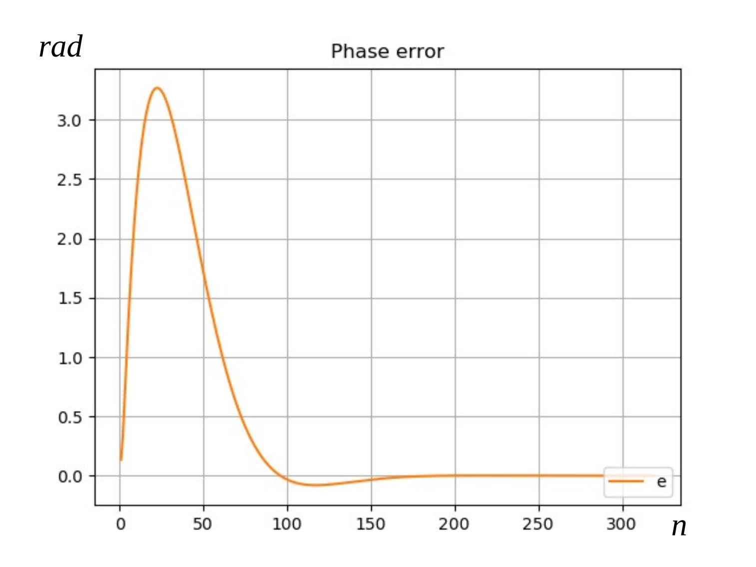
```
>>> from test import *
>>> test=PLLorCostasLoopTest(2e5, 1e4, 1250, 4096, 20, 0.025, 1000,  40.0,
    2e7, lfsel=0, pdtype=0, bmode=1, mu=1.0)
PLLorCostasLoop: Type2 loop filter
>>> test.run( 400000,  4,  65.6, M=1, dteta=0.0, df=8.0, Gs=1.0, Gn=0.0,
    startsw=0, swmode=0, pdmode=0, openloop=0)
BL = 4
phiPM = 65.6
Kp = 11.007002311039455
Ki = 0.031206235560034
cccccccccccccccccccccccccccccccc
n = 400000
k = 320
t = 2.0
exec time = 19.218185663223267
>>> test.show_loop(0,400000)
n rate sample range = 0 : 400000
k rate sample range = 1 : 320
>>> exit()
```

Run Your first test

You shoud see the following diagramms:







PLLpy

Questions