# CARRIER SYNCHRONIZATION

## Software-Defined Radio for Engineers: Chapter 7

Tuesday 13th April, 2021

Manuel Ríos

Universidad Rafael Landívar

# 7.1 Carrier Offsets

# Receiver Blocks

In this chapter we will work with the **Coarse Frequency Correction** (CFC)) and **Carrier Recovery** or **Fine Frequency Correction** (FFC). Matched Filter and Timing Recovery were studied in Chapter 6.

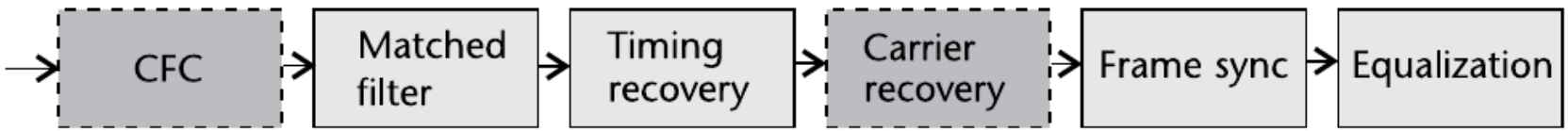


**Figure 1:** Receiver block diagram.

# Introduction

Universidad Rafael Landívar
Tradición Jesuita en Guatemala

Two methods will be discussed:

- Coarse Frequency Correction (CFC)
- Fine Frequency Correction (FFC)

Why Carrier Offset Occurs:

- Receiver and transmitter are distinct and spatially separate units.
- Mismatch between Local Oscillators (LO)
  - Impurities
  - Electrical noise
  - Temperature difference

For simplicity we will model this offset as fixed.

# Frequency Offset

Frequency offset is provided in parts per million (PPM)

$$f_{o,max} = \frac{f_c \times PPM}{10^6}$$

**$f_{o,max}$**

$f_{o,max}$ is important because it provides our carrier recovery design criteria. There is no point wasting resources on a capability to correct for a frequencies beyond our operational range

# Frequency Offset

Mathematically we can model a corrupted source signal at baseband $s(k)$ with a carrier frequency offset of $f_o$ or $\omega_o$ as:

$$r(k) = s(k)e^{j(2\pi f_0 kT+\theta)} + n(k) = s(k)e^{j(\omega_o kT+\theta)} + n(k)$$

Where:

- $n(k)$ is a zero-mean Gaussian random process
- $T$ is the symbol period
- $\theta$ is the carrier phase
- $\omega_0$ the angular frequency

# Estimating the Frequency of a Signal

Universidad Rafael Landívar
Tradición Jesuita en Guatemala

- The frequency of a signal cannot be measured directly unlike the phase.
- By recovering the phase of the signal we are essentially recovering the signal's frequency by the use of:

$$\omega = \frac{d\theta}{dt} = 2\pi f$$

- The instantaneous phase $\theta$ of any complex signal $x(k)$ can be measured as:

$$\theta = \tan^{-1}\left(\frac{\mathrm{Im}(x(k))}{\mathrm{Re}(x(k))}\right)$$

# Estimating the Frequency of a Signal using Matlab

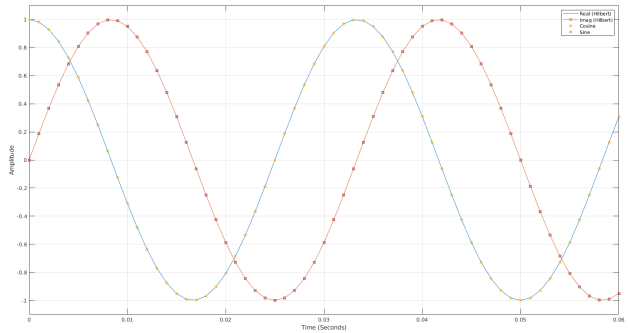


**Figure 2:** Hilbert transform of a real signal

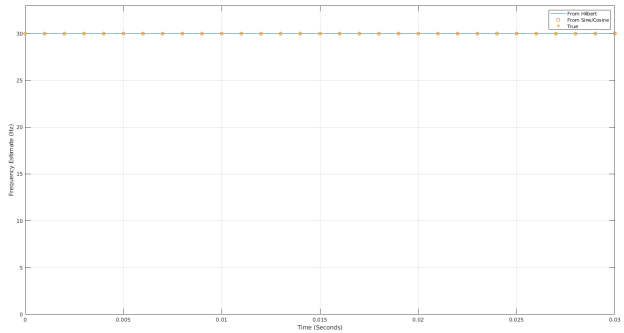


**Figure 3:** Frequency estimation of a signal

Source code: `MATLAB/Chapter_07/freqEstimate.m`

## MATLAB

`hilbert` returns an analytic signal from a real data sequence. `unwrap` prevents phase estimates from becoming bounded between $\pm\pi$

# Frequency Shift of a Signal using Matlab

We can shift a signal in frequency by using:

$$\mathcal{F}\left\{\exp\left(j2\pi f_0 t\right) x(t)\right\} = X\left(f - f_0\right)$$

Source code: `MATLAB/Chapter_07/freqShiftFFT.m`

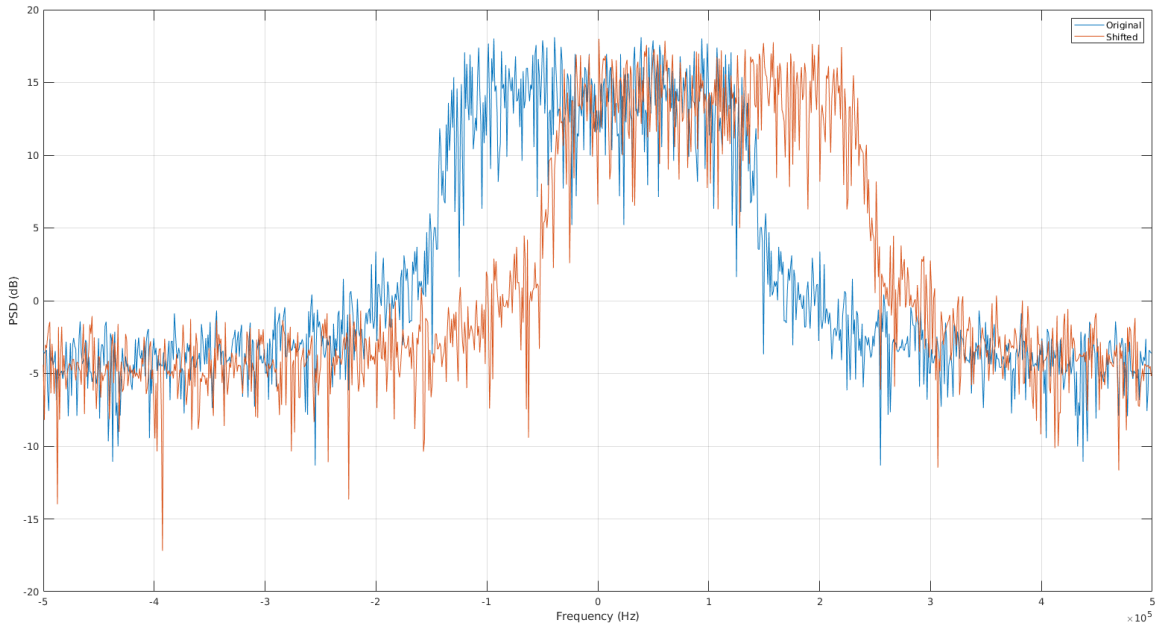


**Figure 4:** Power Spectral Density of BPSK signal with 100-kHz offset.

# 7.2 Frequency Offset Compensation

# Frequency Offset Compensation

Universidad Rafael Landívar
Tradición Jesuita en Guatemala

A two-stage frequency compensation technique:

- Coarse Frequency Correction (CFC)
- Fine Frequency Correction (FFC)

Favorable because:

- Reduces convergence or locking time for estimating the carrier frequency

# 7.2.1 Coarse Frequency Correction

# Coarse Frequency Correction

Universidad Rafael Landívar
Tradición Jesuita en Guatemala

Two primary categories of coarse frequency correction:

- Data-aided (DA)
- Blind or Nondata-aided (NDA)

# Data-aided Correction

- Utilize **correlation type structures** that use knowledge of the received signal, usually in the form of a **preamble**, to estimate the carrier offset $f_o$.
- Performance is generally **limited** by the **length of the preambles**.
- As the **preamble length increases** the system **throughput decreases**.

# Blind or Nondata-aided Correction

- Can operate over the entire duration of the signal.
- Can outperform DA algorithms.
- Typically implemented in an open-loop methodology.
- We will implement a NDA FFT-based technique.

# NDA Coarse Frequency Correction

Universidad Rafael Landívar
Tradición Jesuita en Guatemala

Previously we defined a received signal $r(k)$ as:

$$r(k) = s(k)e^{j(2\pi f_0 kT+\theta)} + n(k)$$

If we neglect the effect of noise we get:

$$r(k) = s(k)e^{j(2\pi f_0 kT+\theta)}$$

We will remove the modulation components of the signal itself by raising the signal to its modulation order $M$:

$$[r(k)]^M = [s(k)e^{j(2\pi f_0 kT+\theta)}]^M$$

# NDA Coarse Frequency Correction

$$[r(k)]^M = [s(k)e^{j(2\pi f_0 kT+\theta)}]^M$$

$$r^M(k) = s^M(k)e^{j(2\pi f_0 kT+\theta)M}$$

- This will shift the offset to $M$ times its original location and make $s(k)$ purely real or purely complex.
- The first term can be ignored and only the remaining exponential or tone will remain.

# NDA Coarse Frequency Correction

- To estimate the position of this tone we will take the FFT of $r^M(t)$ and relate the bin with the most energy to the location of this tone.

$$\hat{f}_o = \frac{1}{2TK} \arg \left| \sum_{k=0}^{K-1} r^M(k) e^{-j2\pi kT/K} \right|$$

# NDA Coarse Frequency Correction using Matlab

Source code: `MATLAB/Chapter_07/freqCoarseEst.m`

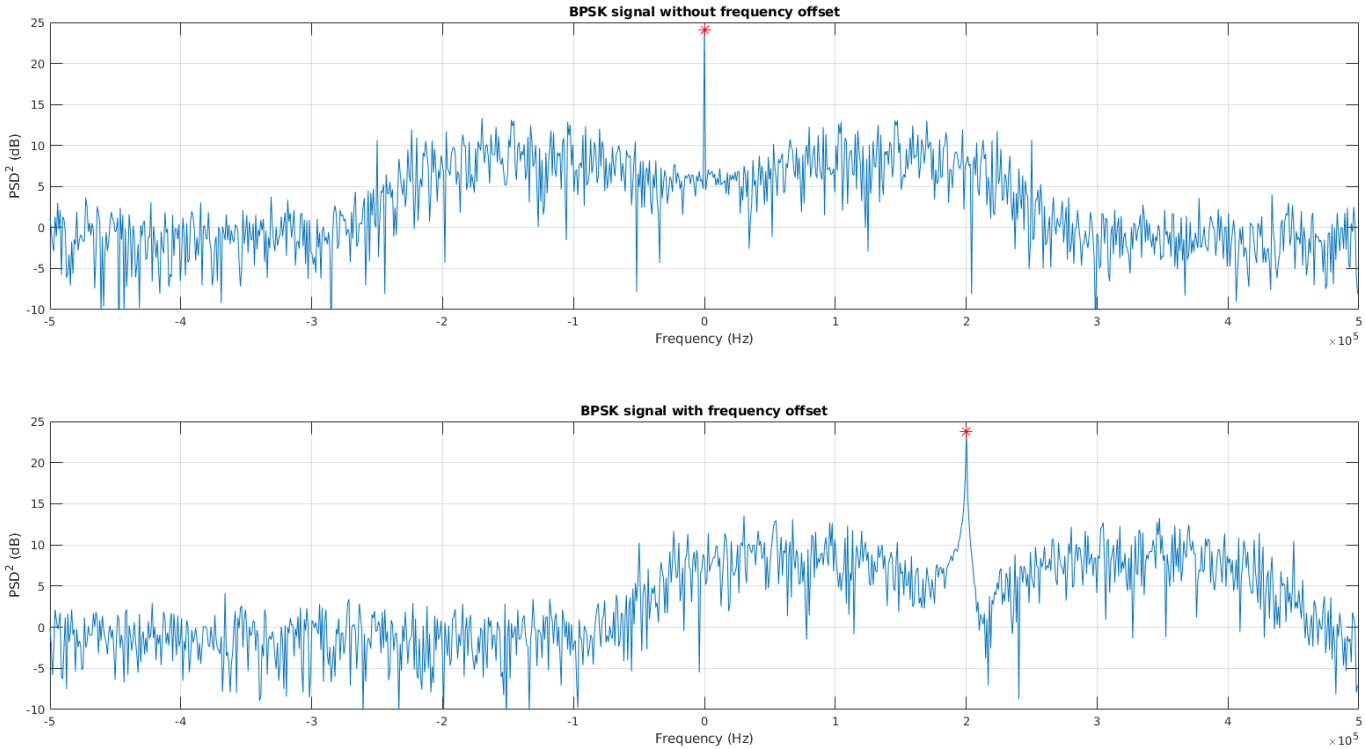


**Figure 5:** Comparison of frequency domain signals with and without frequency offsets

# Why Coarse?

Universidad Rafael Landívar
Tradición Jesuita en Guatemala

Why is it called coarse estimation?

- The resulting $\hat{f}_o$ can only be one of $K$ values produced by the FFT.
- We can extend this accuracy by interpolating across a fixed set of FFT bins over multiple estimates if desired.
- The frequency resolution of each FFT bin for the signal is:

$$f_r = \frac{1}{MTK}$$

  where $M$ is the modulation order, $T$ sampling rate, and $K$ the FFT size.
- Can only be accurate to within $f_r$ ,

# Drawbacks

- From the perspective of downstream algorithms, they will observe a **frequency correction every $K$ samples**. Ideally $f_o$ remains constant, but this is unlikely if the offset is close to an FFT bin boundary. **Frequency jumps** in the signal $\pm f_r$ from previous signals. (Can be smoother with a filter)
- Requires a **significant amount of data** for a reasonable estimate.
- Produce **unpure tones** when oversampled at the transmitter with transmit filters.

# 7.2 Frequency Offset Compensation

## 7.2.2 Fine Frequency Correction

# Fine Frequency Offset

Universidad Rafael Landívar
Tradición Jesuita en Guatemala

- There will still be offset based on the configured resolution chosen $f_r$
- FFC should produce a stable constellation for eventual demodulation
- When a discrete digitally modulated signal exhibits frequency offset, a rotation over time is produced on the constellation
- Positive frequency offset produces a counterclockwise rotation and a negative offset a clockwise rotation.
- The rate of the rotation is equal to the frequency offset.

# Fine Frequency Offset

Source code: `MATLAB/Chapter_07/viewRotation.m`

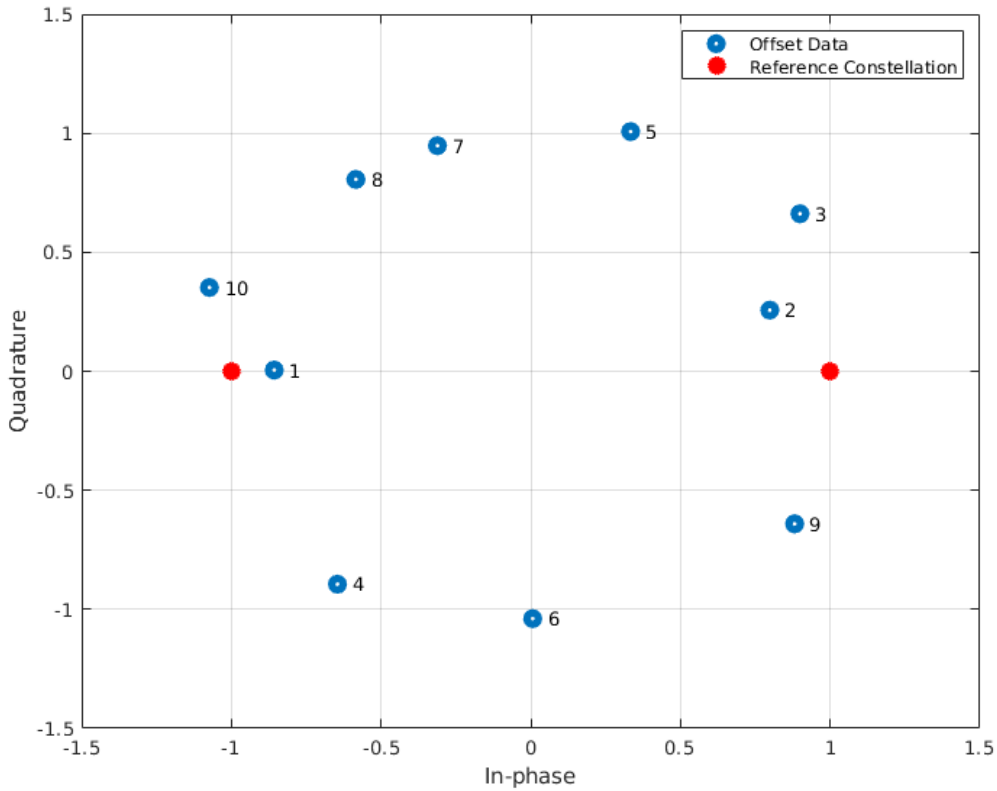


**Figure 6:** Rotating constellation of BPSK source signal with frequency offset.

# Structure of FFC PLL

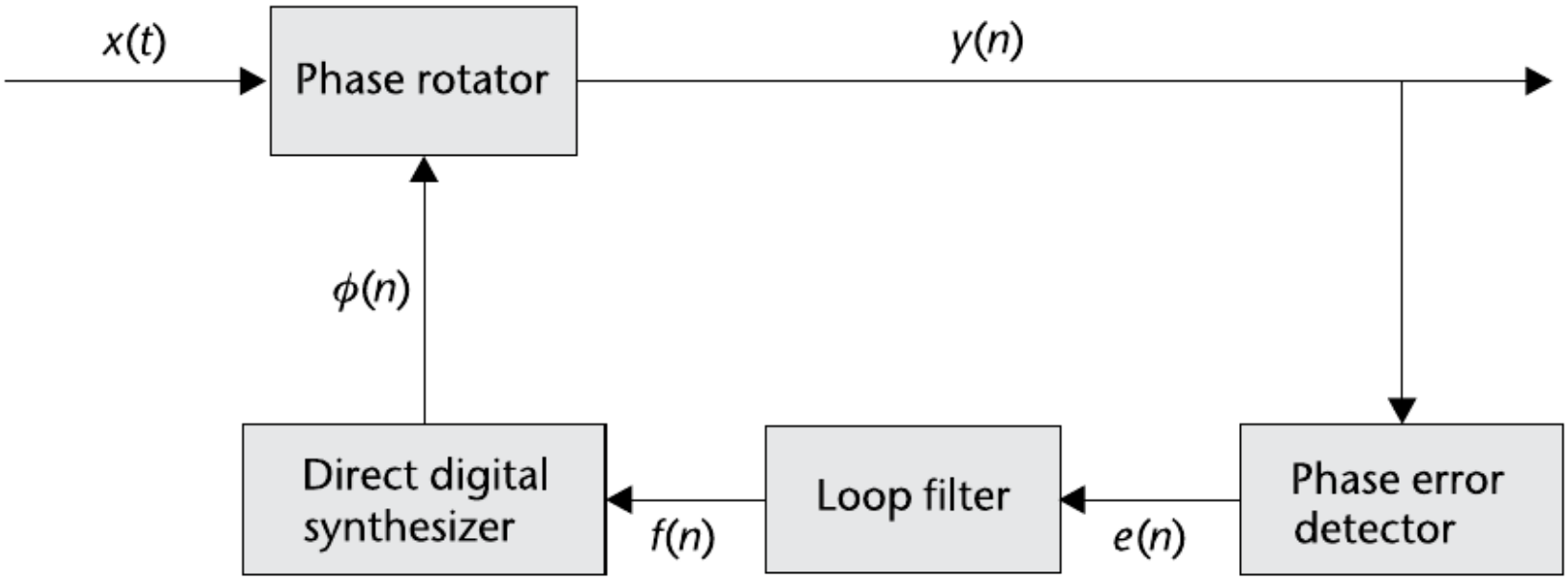


**Figure 7:** Basic PLL structure with four main component blocks.

- **Phase Error Detector**: Produces an error signal $e(n)$ which is based on the structure of the desired receive constellation/symbols.

# Structure of FFC PLL

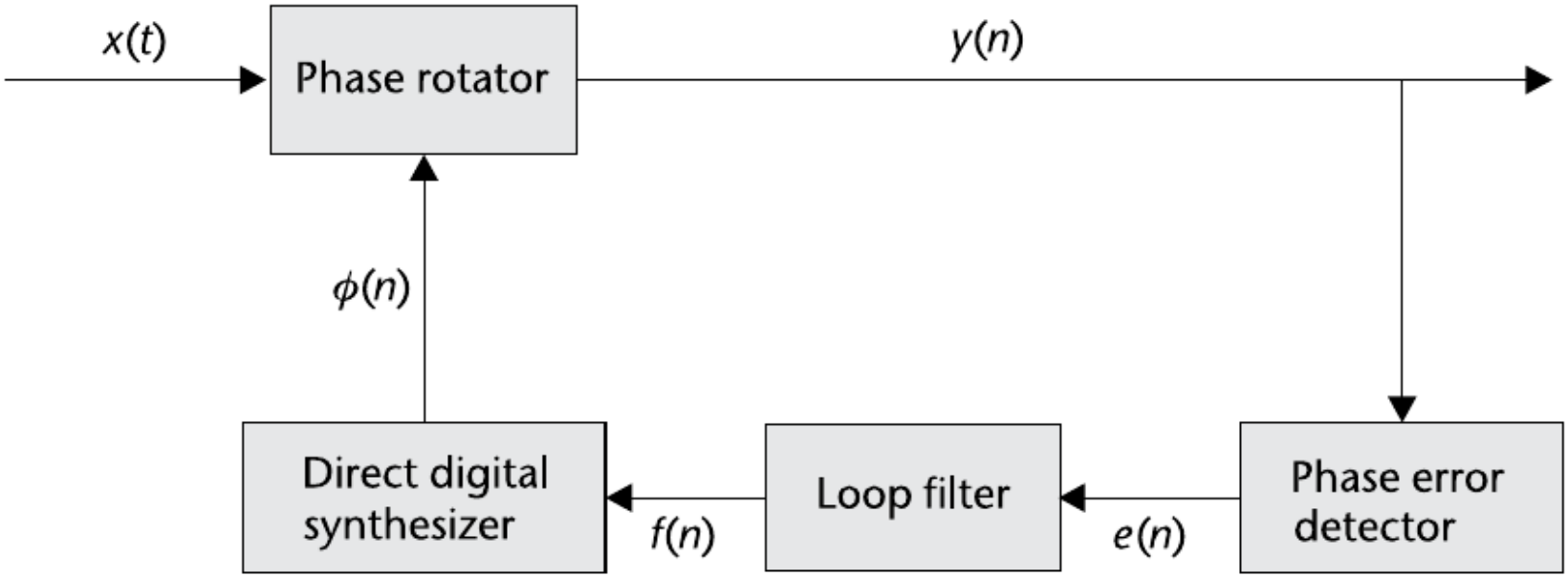


**Figure 8:** Basic PLL structure with four main component blocks.

- **Loop Filter**: Governs the dynamics of the overall PLL. Determines operational frequency, lock time and responsiveness of the PLL, as well as smoothing out the error signal.

# Structure of FFC PLL

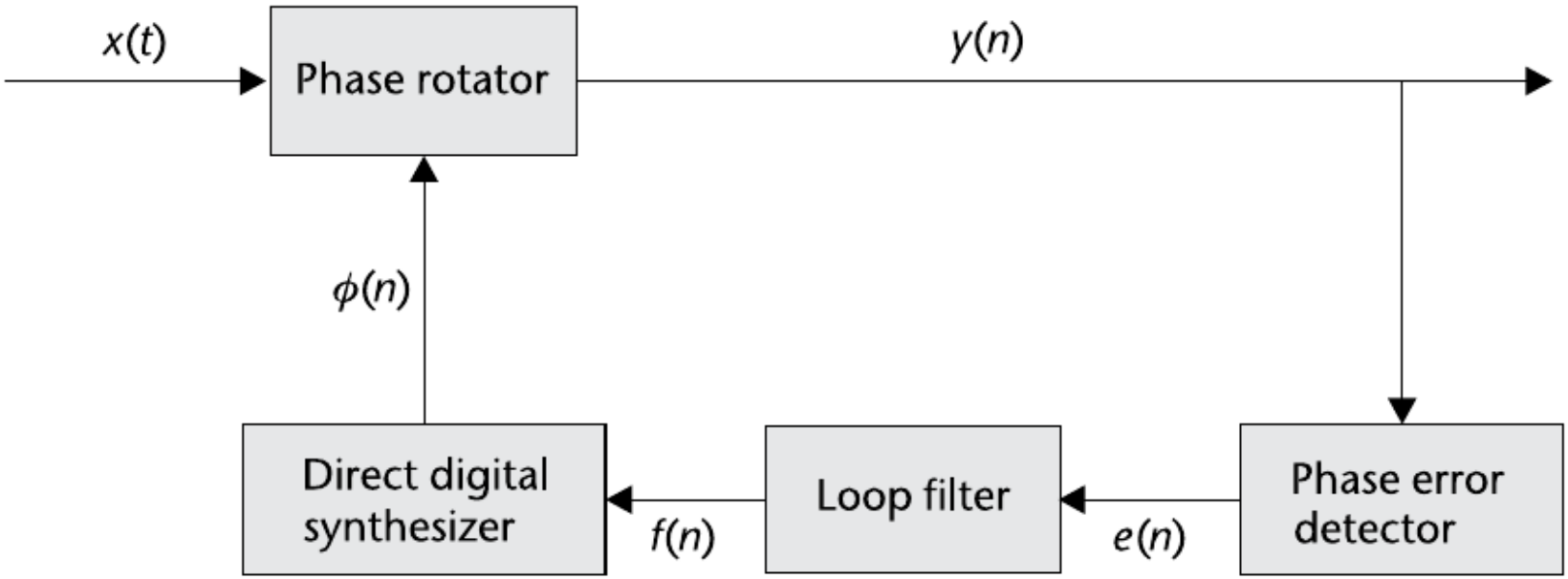


**Figure 9:** Basic PLL structure with four main component blocks.

- **Direct Digital Synthesizer**: Responsible for generation of the correction signal for the input, which again will be fed back into the system.

# Phase Error Detector, PED



- Goal of this block is to **measure** the phase or **radial offset** of the input complex data from a desired reference constellation.
- For QAM, PSK, and PAM $e(n)$ is the **distance from the constellation bases**.
- For FSK or other modulation schemes $e(n)$ can be the **ratio of energy** or **amplitude in a specific basis**.

# Phase Error Detector, PED

Universidad Rafael Landívar
Tradición Jesuita en Guatemala

Error for QPSK signal:

$$e(n) = \text{sign}(\text{Re}(y(n))) \times \text{Im}(y(n)) - \text{sign}(\text{Im}(y(n))) \times \text{Re}(y(n))$$

Error for BPSK or PAM signal:

$$e(n) = \text{sign}(\text{Re}(y(n))) \times \text{Im}(y(n))$$

## Note

If $y(n)$ requires a different orientation this can be accomplished after passing through the synchronizer with a simple multiply with the desired phase shift of:

$$y_{SHIFT}(n) = y(n)e^{j*\phi_{POST}}$$

# Loop Filter -PI Filter

Use a proportional-plus-integrator (PI) filter as our loop filter

$$F(s) = g_1 + \frac{g_2}{s}$$

where $g_1$ and $g_2$ are selectable gains.

With discrete time signals a z-domain representation is preferable:

$$F(z) = G_1 + \frac{G_2}{1 - z^{-}1}$$

where $G_1 \neq g1$ and $G2 \neq g2$. (Proof use bilinear transform)

# Loop Filter -Gain Values

The gain values utilize the following equations based on a preferred damping factor $\zeta$ and loop bandwidth $B_Loop$:

$$\theta = \frac{B_{\text{Loop}}}{M(\zeta + 0.25/\zeta)} \quad \Delta = 1 + 2\zeta\theta + \theta^2$$

$$G_1 = \frac{4\zeta\theta/\Delta}{MK} \quad G_2 = \frac{(4/M)\theta^2/\Delta}{MK}$$

- $M$ is the samples per symbol associated with the input signal.
- $B_Loop$ is a normalized frequency and can range $B_Loop \in [0, 1]$
- $K$ is the detector gain. For QPSK and rectangular QAM $K = 2$, but for PAM and PSK $K = 1$.

# Loop Filter -Damping Factor $\zeta$

For the selection of $\zeta$:

$$\zeta = \begin{cases} < 1, \text{ Underdamp} \\ = 1, \text{ Critically Damped} \\ > 1, \text{ Overdamped} \end{cases}$$

which will determine the responsiveness and stability of the PLL.

- $M$ is the samples per symbol associated with the input signal.
- $B_Loop$ is a normalized frequency and can range $B_Loop \in [0, 1]$

# Loop Filter -Damping Factor $\zeta$

The selection of $B_{Loop}$ should be related to the maximum estimated normalized frequency locking range $\Delta_{f,lock}$ range desired:

$$\Delta_{f,pull} \sim 2\pi\sqrt{2}\zeta B_{Loop}$$

## A good rule of thumb

Start with a damping factor of $\zeta = 1$ and a loop bandwidth of $B_{Loop} = 0.01$. Using an overdamped system here is preferable since it directly increases the pull-in range. However, it will take the loop longer to converge.

# Loop Filter -Damping Factor $\zeta$

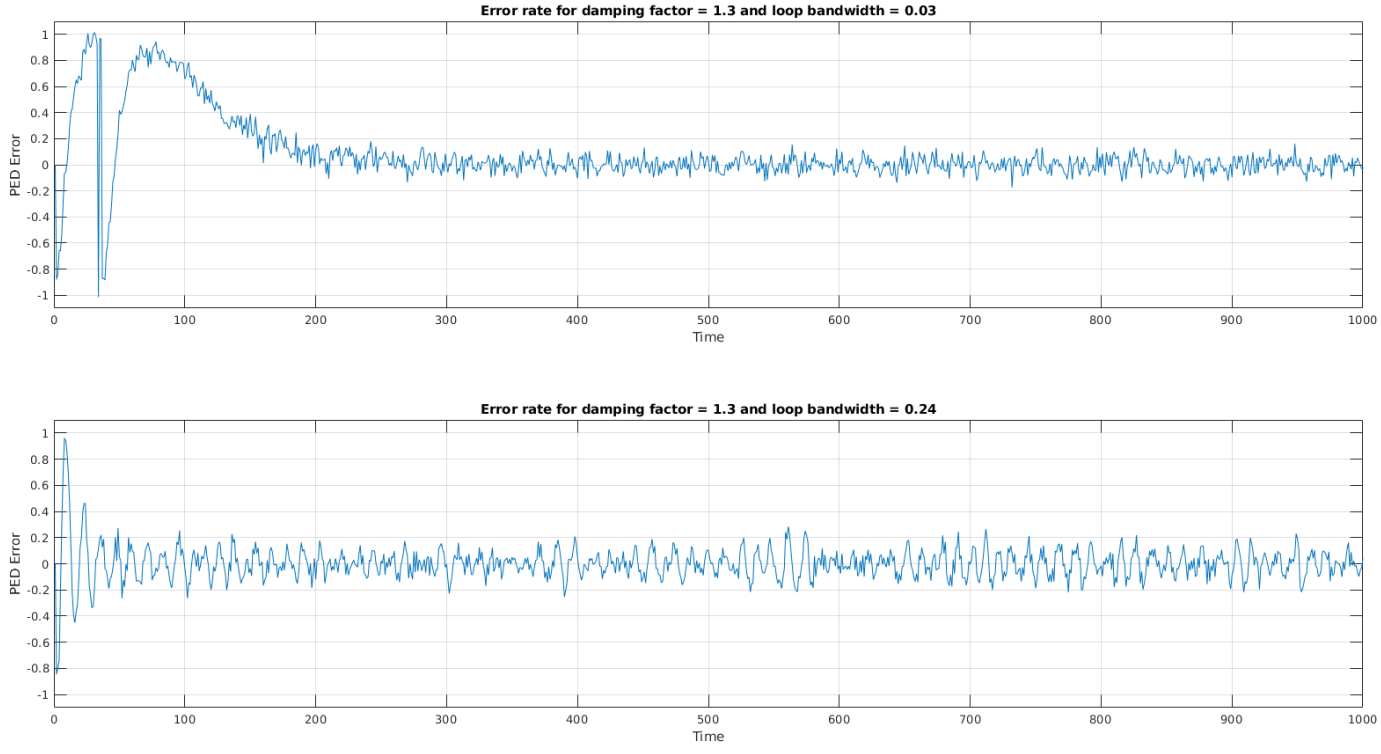


**Figure 10:** Error signal from QPSK PED for different loop bandwidth.

Source code: `MATLAB/Chapter_07/badlock.m`

# Loop Filter -Damping Factor $\zeta$

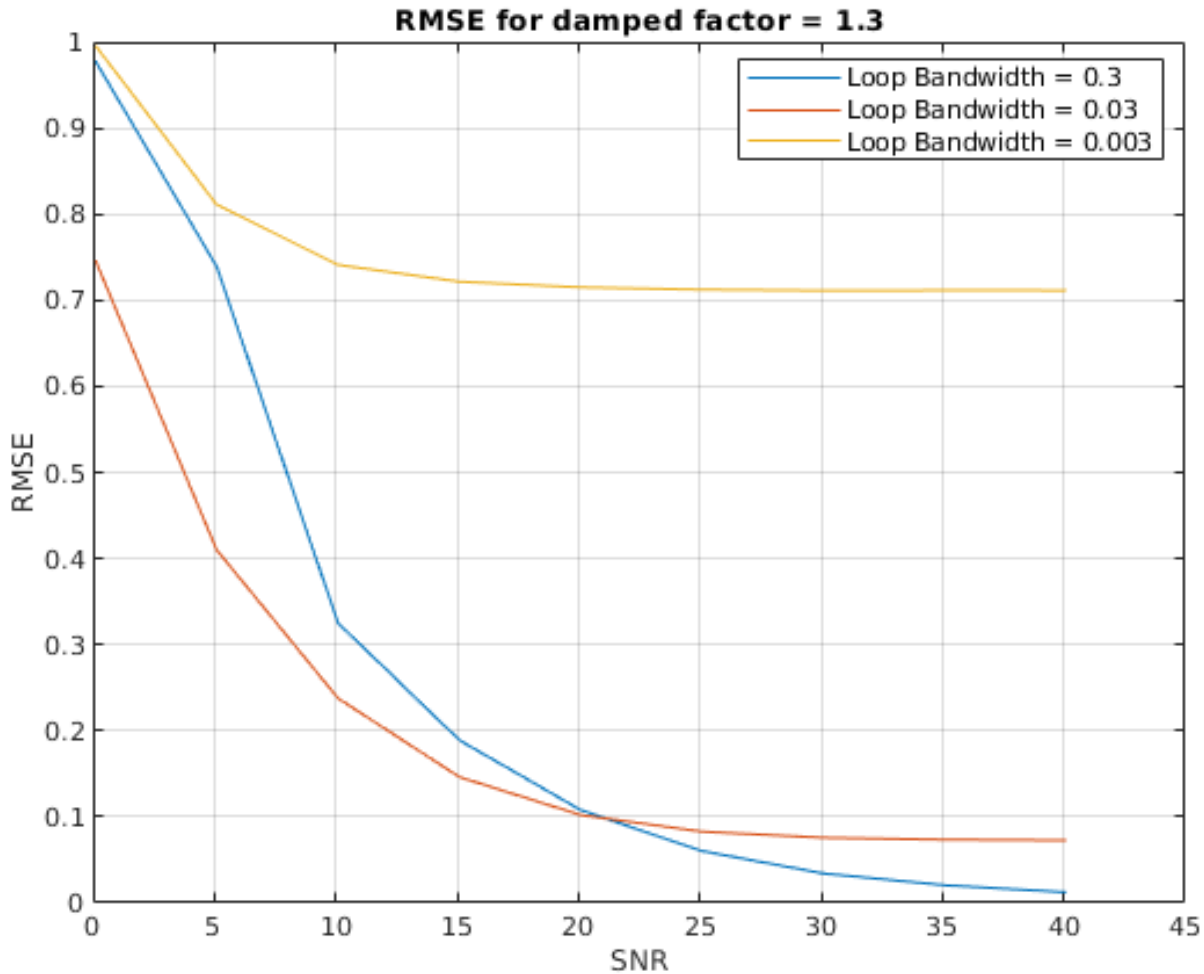


**Figure 11:** SNR vs RMSE of a BPSK signal with fixed damping factor and different loop bandwidths.

# Direct Digital Synthesizer, DDS

Since the loop filter produces a control signal, which is equivalent to the frequency of the input signal, it becomes necessary to extract the phase of this signal instead.

The transfer functions used for the integrator is:

$$D(s) = G_3 \frac{1}{s}$$

Again, a discrete time signals a z-domain representation is preferable:

$$D(z) = G_3 \frac{z^{-1}}{1 - z^{-1}}$$

# Direct Digital Synthesizer, DDS

For equation:

$$D(z) = G_3 \frac{z^{-1}}{1 - z^{-1}}$$

- An additional delay was added to the z-domain representation.
- since we are producing a correction signal $G_3 = -1$.
- This integrator can be implemented with a biquad filter.

**MATLAB**

`dsp.IIRFilter` can be used to implement the required biquad filters for the LF and DDS.

# Direct Digital Synthesizer, DDS

- It can be useful to actually measure the frequency estimation of the synchronizer itself.
- We know that the output of the DDS is the instantaneous phase correction needed for the next symbol, we can simply apply

$$f_{est} = \frac{1}{2\pi}\frac{d\theta}{dt}$$

.

# Direct Digital Synthesizer, DDS

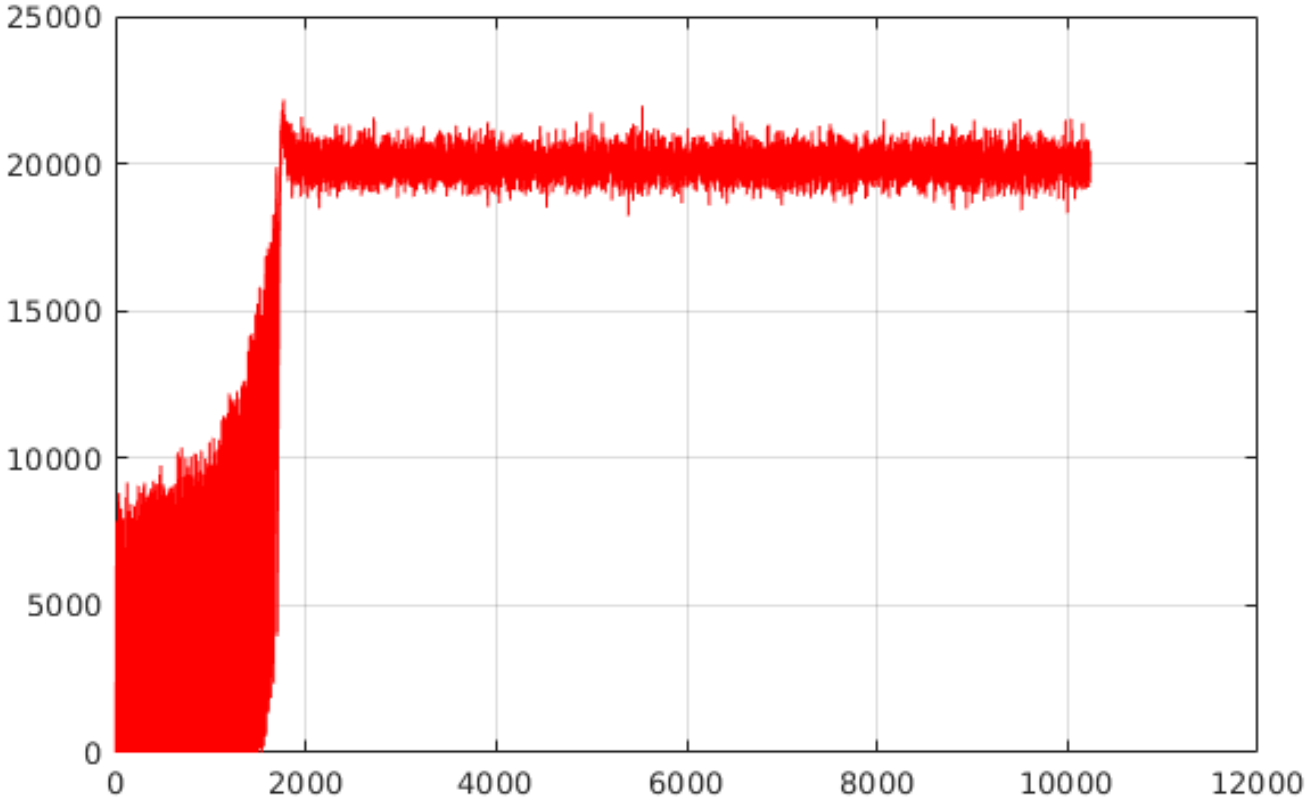


**Figure 12:** Estimations over time and eventual convergence of implemented FFC for 20-KHz offset.

Source code: `MATLAB/Chapter_07/ffcEst.m`

# 7.2 Frequency Offset Compensation

## 7.2.4 Error Vector Magnitude Measurements

# Error Vector Magnitude, EVM

- Provides a measure of **phase noise in the recovered signal**.
- Very useful measurement to **understand the algorithmic performance** in the system.
- Measures the **residual error** of the constellation with respect **to a reference position**.
- Is a **measure** on the **dispersiveness** of the received signal (the lower the EVM values the better).

# Error Vector Magnitude, EVM

We can calculate the EVM in percent RMS as:

$$EVM_{RMS} = 100 \times \sqrt{\frac{\frac{1}{N}\sum_{k=0}^{N-1} e_{const}(k)}{\frac{1}{N}\sum_{k=0}^{N-1} (\mathrm{Re}(\bar{y}(k))^2 + \mathrm{Im}(\bar{y}(k))^2)}}$$

Where:

- $e_{const}(k) = [\mathrm{Re}(y(k)) - \mathrm{Re}(\bar{y}(k))]^2 + [\mathrm{Im}(y(k)) - \mathrm{Im}(\bar{y}(k))]^2$
- $\bar{y}(k)$ is the reference symbol for $y(k)$

There's an error in the book with this equation ¨

# Error Vector Magnitude, EVM

EVM in decibels is very common in OFDM standards due to high-order constellations that can be transmitting, which require a significant EVM margin to recover. EVM in decibels is given by:

$$EVM_{dB} = 20 \log_{10} \left( \frac{EVM_{RMS}}{100} \right)$$

## MATLAB

`comm.EVM` can be used to measure the modulator or demodulator performance of an impaired signal.

Source code: `MATLAB/Chapter_07/estError.m`

# 7.3 Phase Ambiguity

# Phase Ambiguity

- The FFC synchronizer outlined here is blind to the true orientation of the transmitted signal.
- There can be a number of convergent orientations, which can be related to the modulation order.
- Multiple solutions to compensate this problem such as: code words, use of an equalizer with training data, and differential encoding.

# Phase Ambiguity -Code Words

- Relies on a **known sequence** in the received data which is typically in the preamble.
- Demodulating each code word symbol and comparing with the expected result would **provide the necessary mapping** to correctly demodulate the remaining data symbols.
- Demodulate all the preamble symbols and **take the most common orientation** mapping.

# Phase Ambiguity -Differential Encoding

- The goal here is to make the true data dependent on the difference between successive bits.
- To encode the source data we apply the following at the transmitter:

$$b_t(n) = b_t(n-1) \oplus b(n)$$

- To decode the signal we basically perform in reverse as:

$$b(n) = b_t(n) \oplus b_t(n-1)$$

where $b_t$ are the transmitted encoded bits, $b$ are the uncoded bits.

# Phase Ambiguity -Equalizers

There is a whole chapter for this... Chapter 9.

Thank you!!!