

Signals & Systems I

Labs — Code Cheat Sheet (Exam-Ready)

1. Harmonic / Additive Synthesis

```
def synthesize(f0, phi, Ak, t):  
    y = np.zeros_like(t)  
    for k in range(1, len(Ak) + 1):  
        y += Ak[k - 1] * np.cos(  
            2*np.pi*k*f0*t + k*phi - (k-1)*np.pi/2  
        )  
    return y
```

Implements:

$$y(t) = \sum_{k=1}^K A_k \cos\left(2\pi k f_0 t + k\phi - (k-1)\frac{\pi}{2}\right)$$

—

2. FIR Averaging Filter (Moving Average)

```
def averaging_filter(x, N):  
    h = np.ones(N) / N  
    return np.convolve(x, h, mode="same")
```

FIR low-pass smoothing filter.

—

3. FIR Envelope Detector

Rectification followed by FIR low-pass filtering.

```
def envelope(x, N):  
    w = np.abs(x)          # rectification  
    h = np.ones(N) / N     # FIR LPF  
    return np.convolve(w, h, mode="same")
```

—

4. FIR Nulling (Notch) Filter

Removes a sinusoidal interference at frequency f_n .

```
def remove_interference(x, fs, fn=1000):  
    wn = 2*np.pi*fn/fs  
    b = np.array([1.0, -2*np.cos(wn), 1.0])  
    return np.convolve(x, b, mode="same")
```

Implements:

$$y[n] = x[n] - 2 \cos(\omega_n) x[n-1] + x[n-2]$$

—

5. IIR Envelope Detector

Rectification followed by IIR low-pass filtering.

```
def envelope_iir(x, r=0.995):  
    w = np.abs(x)          # rectification  
    G = (1 - r) / 2  
    b = np.array([G, G])  
    a = np.array([1.0, -r])  
    return signal.lfilter(b, a, w)
```

IIR LPF with pole at $z = r$.

—

6. IIR Band-Pass Filter (Resonator)

Extracts a narrow frequency band centered at f_n .

```
def bpf(x, fn, fs, r=0.995):  
    wn = 2*np.pi*fn/fs  
    b = np.array([1.0, 0.0, -1.0])  
    a = np.array([1.0, -2*r*np.cos(wn), r**2])  
    return signal.lfilter(b, a, x)
```

Poles at $re^{\pm j\omega_n}$, zeros at $z = \pm 1$.

—

7. Analysis–Resynthesis with Per-Harmonic Envelopes

(Optional but exam-relevant)

```
def synthesize_with_bpf(x, f0, phi, K, fs, r=0.995):  
    N = len(x)  
    t = np.arange(N) / fs  
    y = np.zeros(N)  
  
    for k in range(1, K + 1):  
        xk = bpf(x, fn=k*f0, fs=fs, r=r)  
        Ak = envelope_iir(xk, r=r)  
        y += Ak * np.cos(  
            2*np.pi*k*f0*t + k*phi - (k-1)*np.pi/2  
        )  
  
    y = y / np.max(np.abs(y)) * np.max(np.abs(x))  
    return y
```

Pipeline:

$$x \rightarrow \text{BPF}_{kf_0} \rightarrow \text{Envelope} \rightarrow A_k[n]$$