# 🎯 Signal Customization & Tuning Guide

## Why Customize Signals?

Every trader has:

- Different risk tolerance
- Different trading style (scalping vs swing)
- Different capital size
- Different market understanding

**This guide shows you how to tune the signal engine to YOUR trading personality.**

---

## 🔧 Signal Components Breakdown

### Current Scoring System (Default)

```python
Component              Points   What It Measures
─────────────────────────────────────────────────────
VWAP Alignment           20       Price structure
OI Unwinding             25     Support/Resistance weakness
OI Addition (Opposite)   20       New directional conviction
Max Pain Distance        15      Expansion potential
Time Window              10     Optimal trading hours
Volume Expansion         10      Momentum confirmation
─────────────────────────────────────────────────────
TOTAL                   100
```

### Decision Thresholds:

- 75-100: BUY 🟢
- 50-74: WAIT 🟡
- 0-49: AVOID 🔴

---

## 🎨 Customization Scenarios

### Scenario 1: Conservative Trader (Higher Win Rate)

**Profile:**

- Capital: ₹50,000 - ₹2,00,000
- Goal: 60%+ win rate, smaller losses
- Style: Patient, takes fewer trades

**Modifications:**

python
```python
# 1. Increase BUY threshold
if score >= 80:  # Changed from 75
    signal_type = "BUY"
elif score >= 60:  # Changed from 50
    signal_type = "WAIT"
else:
    signal_type = "AVOID"
```

python
```python
# 2. Add stricter time filter
def check_time_window(self):
    current_hour = datetime.now().hour
    current_minute = datetime.now().minute

    # Only 12:00 PM - 2:00 PM (most stable period)
    if 12 <= current_hour < 14:
        return 15  # Increased points
    else:
        return -10  # Penalty for wrong time
```

python
```python
# 3. Require higher OI Change
if abs(row['Call OI Change']) > 15000:  # Changed from 10000
    score += 25
```

**Expected Results:**

- Fewer signals (3-5 per week vs 8-10)
- Win rate: 65-70%

- Smaller drawdowns

---

**Scenario 2: Aggressive Trader (More Opportunities)**

**Profile:**

- Capital: ₹5,00,000+
- Goal: More trades, faster profits
- Style: Active, comfortable with volatility

**Modifications:**

```python
# 1. Lower BUY threshold
if score >= 65:  # Changed from 75
    signal_type = "BUY"
elif score >= 45:  # Changed from 50
    signal_type = "WAIT"
```

```python
# 2. Expand time window
if 10 <= current_hour < 15:  # Changed from 11-15
    score += 10
```

```python
# 3. Add momentum multiplier
def calculate_momentum_boost(self):
    """Reward strong directional moves"""
    if abs(self.spot_price - self.max_pain) > 1.0:  # >1% away
        return 15  # Bonus points
    return 0
```

**Expected Results:**

- More signals (12-15 per week)
- Win rate: 55-60%
- Higher volatility in returns

---

**Scenario 3: Scalper (Intraday Only)**

**Profile:**

- Capital: ₹2,00,000+
- Goal: Quick 15-30 point moves
- Style: In and out within 30 mins

**Modifications:**

```python
# 1. Focus on OI Change ONLY (ignore static OI)
def generate_signal(self):
    score = 0

    # OI Change gets 50 points (50% weight)
    call_oi_change = self.df['Call OI Change'].sum()
    put_oi_change = self.df['Put OI Change'].sum()

    if abs(call_oi_change) > 20000 or abs(put_oi_change) > 20000:
        score += 50
```

```python
# 2. Add volume surge detection
def check_volume_surge(self):
    """Look for sudden volume spikes"""
    # Compare current volume to 15-min average
    if current_volume > avg_volume * 1.5:
        return 20  # High confidence
    return 0
```

```python
# 3. Ignore Max Pain completely
# Comment out or set weight to 0
# Max Pain is irrelevant for 30-min scalps
```

**Expected Results:**

- High frequency signals (5-8 per day)
- Win rate: 52-58%
- Quick exits required

---

**Scenario 4: Swing Trader (Hold 1-3 Days)**

**Profile:**

- Capital: ₹3,00,000+
- Goal: Larger moves (100-200 points)
- Style: Position trading

**Modifications:**

```python
# 1. Increase Max Pain weight
def calculate_max_pain_score(self):
    distance_pct = abs(self.spot_price - self.max_pain) / self.spot_price * 100

    if distance_pct > 0.5:  # Changed from 0.35
        return 30  # Increased from 15
    return 0
```

```python
# 2. Add trend confirmation
def check_multi_day_trend(self):
    """Require 2-3 day OI build-up"""
    # Check if OI has been building consistently
    # (Requires historical data)
    if oi_trending_for_days >= 2:
        return 20
    return -10  # Penalty if no trend
```

```python
# 3. Reduce time sensitivity
# Remove time window check completely
# Swing trades can be entered anytime
```

**Expected Results:**

- Fewer signals (2-4 per week)
- Win rate: 60-65%
- Larger profit targets

## 🎛️ Advanced Tuning Parameters

### Parameter 1: OI Change Sensitivity

```python
# Location: In generate_signal() method

# Default
if total_call_oi_change > 0:
    score += 20

# Conservative (require more OI)
if total_call_oi_change > 15000:  # Specific threshold
    score += 20

# Aggressive (any OI change counts)
if total_call_oi_change != 0:
    score += 20
```

---

### Parameter 2: PCR Interpretation

```python
# Location: In detect_market_type() method

# Default
if 0.9 <= self.pcr <= 1.1:
    return "RANGE"

# Conservative (wider range = more AVOID signals)
if 0.85 <= self.pcr <= 1.15:
    return "RANGE"

# Aggressive (narrower range = more BUY signals)
if 0.95 <= self.pcr <= 1.05:
    return "RANGE"
```

---

### Parameter 3: Distance from Max Pain

```python
```

```
# Location: In generate_signal() method

# Default
if distance_pct > 0.35:
    score += 15

# Conservative (require further distance)
if distance_pct > 0.5:
    score += 20

# Aggressive (accept closer proximity)
if distance_pct > 0.25:
    score += 15
```

---

## 🧪 Testing Your Custom Settings

### Step 1: Create Test Scenarios

```
python
```

```python
# test_signals.py

test_scenarios = [
    {
        'name': 'Strong Bullish',
        'spot': 59500,
        'max_pain': 59200,
        'pcr': 0.8,
        'call_oi_change': 50000,
        'put_oi_change': -20000,
        'expected_signal': 'BUY'
    },
    {
        'name': 'Range Bound',
        'spot': 59300,
        'max_pain': 59280,
        'pcr': 1.0,
        'call_oi_change': 5000,
        'put_oi_change': 5000,
        'expected_signal': 'AVOID'
    },
    # Add more scenarios
]

def test_custom_signals():
    for scenario in test_scenarios:
        # Create test DataFrame
        df = create_test_df(scenario)

        # Generate signal
        analyzer = OptionChainAnalyzer(df)
        result = analyzer.generate_signal()

        # Check result
        if result['signal'] == scenario['expected_signal']:
            print(f"✅ {scenario['name']}: PASS")
        else:
            print(f"❌ {scenario['name']}: FAIL")
            print(f"   Expected: {scenario['expected_signal']}")
            print(f"   Got: {result['signal']}")
```

---

**Step 2: Paper Trade First**

**Before going live with custom settings:**

1. Run signals on historical data for 1 week
2. Track hypothetical P&L
3. Note emotional reactions to signals
4. Adjust if needed
5. Repeat for 2-3 weeks

**Tracking Template:**

```csv
Date,Time,Signal,Strike,Entry,Exit,P&L,Notes
2024-01-15,11:45,BUY CALLS,59500,145,162,+17,"Good entry, weak exit"
2024-01-15,14:20,AVOID,-,-,-,-,"Correctly avoided chop"
```

---

## 🎯 Signal Quality Metrics

### Track These KPIs

```python
```

```
class SignalMetrics:
    def __init__(self):
        self.total_signals = 0
        self.buy_signals = 0
        self.avoid_signals = 0
        self.wins = 0
        self.losses = 0

    def calculate_quality(self):
        """
        Signal Quality Score (0-100)
        """
        win_rate = self.wins / self.buy_signals if self.buy_signals > 0 else 0
        avoid_accuracy = self.correct_avoids / self.avoid_signals if self.avoid_signals > 0 else 0

        # Ideal: 65% win rate, 80% avoid accuracy
        quality_score = (win_rate * 0.6 + avoid_accuracy * 0.4) * 100

        return {
            'quality_score': quality_score,
            'win_rate': win_rate * 100,
            'avoid_accuracy': avoid_accuracy * 100,
            'total_signals': self.total_signals,
            'profitability': (self.wins - self.losses) / self.total_signals * 100
        }
```

**Good Metrics:**

- Quality Score: >70
- Win Rate: >60%
- Avoid Accuracy: >75%
- Profitability: Positive

---

## 🔄 Dynamic Adjustment Strategy

Instead of fixed thresholds, adapt to market conditions:

```
python
```

```
class AdaptiveSignalEngine:
    def __init__(self):
        self.volatility_regime = "NORMAL"  # LOW, NORMAL, HIGH

    def detect_volatility_regime(self):
        """Adjust thresholds based on market volatility"""
        # Calculate ATR or use VIX proxy

        if vix_equivalent > 25:
            self.volatility_regime = "HIGH"
            self.buy_threshold = 80  # Stricter in high vol
        elif vix_equivalent < 15:
            self.volatility_regime = "LOW"
            self.buy_threshold = 70  # More relaxed in low vol
        else:
            self.volatility_regime = "NORMAL"
            self.buy_threshold = 75

    def generate_adaptive_signal(self):
        self.detect_volatility_regime()

        # Use dynamic threshold
        if score >= self.buy_threshold:
            return "BUY"
        # ... rest of logic
```

---

## 📊 A/B Testing Framework

Run two configurations simultaneously:

```
python
```

```
# Config A: Conservative
config_a = {
    'buy_threshold': 80,
    'pcr_range': (0.85, 1.15),
    'time_window': (12, 14)
}

# Config B: Aggressive
config_b = {
    'buy_threshold': 65,
    'pcr_range': (0.9, 1.1),
    'time_window': (10, 15)
}

# Track results separately
results_a = []
results_b = []

# After 2 weeks, compare:
# - Win rate
# - Total signals
# - Max drawdown
# - Profit factor
```

---

## 💡 Pro Tips for Signal Tuning

### 1. Start Conservative, Go Aggressive Later

Begin with high thresholds (80+). As you gain confidence, lower them.

### 2. Tune One Parameter at a Time

Don't change 5 things at once. You won't know what worked.

### 3. Market Conditions Matter

Settings that work in trending markets may fail in range-bound periods.

### 4. Document Everything

Keep a journal of:

- What you changed
- Why you changed it
- Results after 1 week

**5. Accept Imperfection**

No signal system is 100% accurate. Target 60-65% win rate.

---

## 🎓 Signal Tuning Checklist

Before deploying custom settings:

- [ ] Tested on at least 20 historical scenarios
- [ ] Paper traded for minimum 2 weeks
- [ ] Win rate above 55%
- [ ] Avoid accuracy above 70%
- [ ] Comfortable with false positives
- [ ] Understand why each parameter exists
- [ ] Have rollback plan if settings fail
- [ ] Documented all changes
- [ ] Set realistic profit targets
- [ ] Defined maximum loss per trade

---

## 🆘 When to Reset to Defaults

**Reset if you see:**

- Win rate drops below 45% for 2 weeks
- More than 5 consecutive losses
- Signal frequency too high/low for your comfort
- You're second-guessing every signal
- Emotional stress increases

**Default settings exist for a reason—they work for most traders.**

---

## 🔮 Future: Machine Learning Optimization

Eventually, you can use ML to auto-tune:

```
python
```

```
from sklearn.ensemble import RandomForestClassifier

# Features: score, pcr, oi_change, time, etc.
# Target: actual_profit_loss

model = RandomForestClassifier()
model.fit(X_train, y_train)

# Get optimal thresholds
optimal_threshold = model.feature_importances_
```

But start manual first—understand the logic before automating.

---

## 📚 Recommended Reading Order

1. Use defaults for 1 month
2. Identify pain points (too many signals? too few?)
3. Read relevant scenario above
4. Change ONE parameter
5. Test for 2 weeks
6. Iterate

---

**Remember:** The goal isn't perfect signals. It's signals that match YOUR trading style and psychology.

Good luck tuning! 🎯