PyForecast Refinement Guide

Fit Quality Analysis and Parameter Learning

January 2026

PyForecast Refinement Guide

Fit Quality Analysis and Parameter Learning

Date: January 2026

Executive Summary

PyForecast includes a refinement module for measuring, logging, analyzing, and improving decline curve fit quality. This system provides:

* **Hindcast Validation** - Measure actual forecast accuracy through backtesting
* **Fit Logging** - Persist all fit parameters and metrics for analysis
* **Residual Analysis** - Detect systematic fit errors (bias, autocorrelation)
* **Parameter Learning** - Suggest optimal parameters from historical performance
* **Ground Truth Comparison** - Compare auto-fitted curves against expert ARIES projections

Key Design Principle: All refinement features are disabled by default and observation-only. They don't modify the core fitting behavior-they observe and report on it.

Quick Start

Enable Refinement Features via CLI

```
# Run with hindcast validation
pyforecast process data.csv --hindcast -o output/

# Run with fit logging
pyforecast process data.csv --log-fits -o output/

# Run with residual analysis
pyforecast process data.csv --residuals -o output/

# Enable all refinement features
pyforecast process data.csv --hindcast --log-fits --residuals -o output/
```

Enable via Configuration

```
# pyforecast.yaml
refinement:
  enable_logging: true
  log_storage: sqlite
  log_path: null  # Uses ~/.pyforecast/fit_logs.db

  enable_hindcast: true
  hindcast_holdout_months: 6
  min_training_months: 12

  enable_residual_analysis: true

  enable_learning: false
```

Feature 1: Hindcast Validation

Hindcast (backtesting) splits historical data into training and holdout periods, fits on training data, and measures prediction accuracy on holdout.

How It Works

```
Historical Data (24 months)
|-- Training Period (18 months) ? Fit decline curve
\-- Holdout Period (6 months)  ? Measure prediction accuracy
```

1. Check data sufficiency: `n_months >= min_training + holdout`
2. Split data: training = first (n - holdout) months
3. Fit on training data only
4. Predict holdout period using fitted model
5. Compute accuracy metrics

Metrics Computed

Metric	Description	Good Value
MAPE	Mean Absolute Percentage Error	< 30%
Correlation	Pearson correlation coefficient	> 0.7
Bias	Systematic over/under prediction	|bias| < 0.3

CLI Usage

```
# Run hindcast validation
pyforecast process data.csv --hindcast -o output/

# Output includes:
# - output/refinement_report.txt with hindcast summary
# - Per-well MAPE, correlation, and bias
```

Example Output

```
Refinement Analysis:
  Hindcast validation: 45 wells
    Average MAPE: 18.5%
    Good hindcast rate: 78.0%

  See output/refinement_report.txt for details
```

Configuration

```
refinement:
  enable_hindcast: true
  hindcast_holdout_months: 6    # Months to hold out
  min_training_months: 12       # Minimum training data
```

Programmatic Usage

```
from pyforecast.refinement import HindcastValidator
from pyforecast.core.fitting import DeclineFitter

validator = HindcastValidator(holdout_months=6, min_training_months=12)
fitter = DeclineFitter()

# Validate a well
result = validator.validate(well, "oil", fitter)

if result:
    print(f"MAPE: {result.mape:.1f}%")
    print(f"Correlation: {result.correlation:.3f}")
    print(f"Good hindcast: {result.is_good_hindcast}")
```

Feature 2: Fit Logging

Fit logging persists all fit parameters, metrics, and diagnostics to SQLite storage for analysis and learning across projects.

Storage Location

Default: `~/.pyforecast/fit_logs.db`
This cumulative database enables learning from all fits over time.

What Gets Logged

Category	Fields
Identification	fit_id, timestamp, well_id, product
Context	basin, formation (if available)
Input	data_points_total, data_points_used, regime_start_idx
Parameters	b_min, b_max, dmin_annual, recency_half_life, regime_threshold
Results	qi, di, b, r_squared, rmse, aic, bic
Residuals	residual_mean, durbin_watson, early_bias, late_bias
Hindcast	hindcast_mape, hindcast_correlation, hindcast_bias

CLI Usage

```
# Enable fit logging during processing
pyforecast process data.csv --log-fits -o output/

# Analyze accumulated fit logs
pyforecast analyze-fits --basin "Permian" -o analysis.csv

# View fit log statistics
pyforecast analyze-fits ~/.pyforecast/fit_logs.db
```

Analyze Fits Command

```
pyforecast analyze-fits [STORAGE_PATH] [OPTIONS]

Options:
  --basin TEXT      Filter by basin name
  --formation TEXT  Filter by formation name
  -p, --product TEXT  Filter by product
  -o, --output PATH   Export analysis to CSV
```

Example Output

```
Analyzing fit logs from ~/.pyforecast/fit_logs.db

Fit Log Summary:
  Total fits: 523
  Average R?: 0.892
  Average RMSE: 45.32

Hindcast Performance:
  Fits with hindcast: 485
  Average MAPE: 17.3%
  Good hindcast rate: 82.1%

B-Factor Distribution:
  Mean: 0.524
  Std: 0.312
  Range: [0.010, 1.500]
```

Programmatic Usage

```
from pyforecast.refinement import FitLogger, FitLogAnalyzer

# Log fits during processing
with FitLogger() as logger:
    for well in wells:
        result = fitter.fit(t, q)
        logger.log(result, well, "oil", fitting_config)

# Analyze logged fits
analyzer = FitLogAnalyzer()
summary = analyzer.get_summary(basin="Permian", product="oil")
print(f"Average R?: {summary['avg_r_squared']:.3f}")
```

Feature 3: Residual Analysis

Residual analysis detects systematic fit errors that may not be apparent from R? alone.

Diagnostics Computed

Diagnostic	Description	Ideal Value
Durbin-Watson	Autocorrelation test	~2.0
Early Bias	Error in first half of data	~0%
Late Bias	Error in last half of data	~0%
Autocorr Lag-1	Lag-1 autocorrelation	~0

Durbin-Watson Interpretation

DW Value	Interpretation
< 1.5	Positive autocorrelation (systematic underfitting)
1.5 - 2.5	No significant autocorrelation (good fit)
> 2.5	Negative autocorrelation (overfitting/oscillation)

Validation Issues Generated

Code	Severity	Description
RD001	WARNING	Significant autocorrelation in residuals
RD002	WARNING	Systematic early/late bias pattern
RD003	INFO	Non-zero mean residuals

CLI Usage

```
# Enable residual analysis
pyforecast process data.csv --residuals -o output/

# Output includes residual diagnostics in validation_report.txt
```

Example Output

```
Residual Analysis Summary:
  Total analyzed: 45
  With systematic patterns: 8 (17.8%)

Wells with systematic residual patterns:
  WELL-001/oil: DW=1.23, early_bias=-12.5%, late_bias=8.3%
  WELL-015/oil: DW=0.98, early_bias=15.2%, late_bias=-5.1%
```

Programmatic Usage

```
from pyforecast.refinement import ResidualAnalyzer
from pyforecast.refinement.schemas import ResidualDiagnostics

analyzer = ResidualAnalyzer()

# Analyze residuals
diagnostics = ResidualDiagnostics.compute(actual, predicted)

print(f"Durbin-Watson: {diagnostics.durbin_watson:.2f}")
print(f"Early bias: {diagnostics.early_bias:.1%}")
print(f"Late bias: {diagnostics.late_bias:.1%}")
print(f"Has systematic pattern: {diagnostics.has_systematic_pattern}")

# Get validation issues
validation_result = analyzer.get_validation_issues(diagnostics, "well_id", "oil")
for issue in validation_result.issues:
    print(f"[{issue.code}] {issue.message}")
```

Feature 4: Parameter Learning

Parameter learning analyzes accumulated fit logs to suggest optimal fitting parameters for different basins and formations.

Parameters That Can Be Learned

Parameter	Description
`recency_half_life`	How aggressively to weight recent data
`regime_threshold`	What increase triggers regime detection
`regime_window`	Months of trend data for regime detection
`regime_sustained_months`	How long elevation must persist

Note: Physical parameters (b_min, b_max, dmin) are NOT learned-they're literature-based constraints.

How Learning Works

1. Accumulate fit logs with hindcast data
2. Group by basin/formation (or global)
3. Weight fits by hindcast performance (lower MAPE = higher weight)
4. Derive optimal parameter values
5. Store suggestions in database

Confidence Levels

Sample Count	Confidence
< 20	Low
20 - 99	Medium
? 100	High

CLI Usage

```
# Get parameter suggestions
pyforecast suggest-params -p oil --basin "Permian"

# Update all suggestions from current fit logs
pyforecast suggest-params --update

# Export suggestions to CSV
pyforecast suggest-params -o suggestions.csv
```

Example Output

```
Parameter Suggestion for Permian/oil
  Based on 156 fits (confidence: high)
```

```
Suggested values:
  recency_half_life: 10.5
  regime_threshold: 0.85
  regime_window: 6
  regime_sustained_months: 2

Historical performance with these parameters:
  Average R?: 0.912
  Average hindcast MAPE: 14.2%
```

Programmatic Usage

```
from pyforecast.refinement import ParameterLearner
from pyforecast.refinement.parameter_learning import apply_suggestion

learner = ParameterLearner()

# Get suggestion for a basin
suggestion = learner.suggest(product="oil", basin="Permian")

if suggestion:
    print(f"Suggested half-life: {suggestion.suggested_recency_half_life}")
    print(f"Confidence: {suggestion.confidence}")

    # Apply suggestion to create new config
    new_config = apply_suggestion(suggestion, base_fitting_config)
```

Feature 5: Ground Truth Comparison

Ground truth comparison measures how well pyforecast's auto-fitted decline curves match expert/approved ARIES projections.

Use Case

When you have existing ARIES forecasts created by reservoir engineers, you can compare pyforecast's automated fits against these "ground truth" projections to:

* Validate that auto-fitting produces reasonable results
* Identify wells where manual review may be needed
* Measure overall fitting accuracy across a portfolio

ARIES Expression Format

PyForecast parses ARIES AC_ECONOMIC expression format:

```
"{Qi} X {unit} {Dmin%} {type} B/{b} {Di%}"
```

Example: `"1000 X B/M 6 EXP B/0.50 8.5"` means:

* qi = 1000 bbl/month
* unit = B/M (barrels/month)
* dmin = 6% annual terminal decline
* type = EXP (exponential)
* b = 0.50 hyperbolic exponent
* di = 8.5% annual initial decline

CLI Usage

```
# Run with ground truth comparison
pyforecast process data.csv --ground-truth aries_projections.csv -o output/

# With custom comparison period (default: 60 months)
pyforecast process data.csv --ground-truth aries_projections.csv --gt-months 120 -o output/

# Generate comparison plots for each well
pyforecast process data.csv --ground-truth aries_projections.csv --gt-plots -o output/

# Use lazy loading for large ARIES files (streams instead of loading into memory)
pyforecast process data.csv --ground-truth large_aries.csv --gt-lazy -o output/

# Enable parallel validation for large batches
pyforecast process data.csv --ground-truth aries_projections.csv --gt-workers 4 -o output/
```

Example Output

```
Ground Truth Comparison:
  Wells with ARIES data: 45 of 50
```

```
  Average MAPE: 12.3%
  Average correlation: 0.987
  Good match rate: 82.2%

  See output/ground_truth_report.txt for details
```

Metrics Computed

Metric	Description	Good Value
MAPE	Mean Absolute Percentage Error of rates	< 20%
Correlation	Pearson correlation of rate curves	> 0.95
Cumulative Diff	Difference in total production	< 15%
B-Factor Diff	Absolute difference in b-factor	< 0.3

Match Quality Grades

Wells are graded A-D based on overall match quality:

Grade	Criteria
A	Excellent match - all metrics well within thresholds
B	Good match - minor deviations
C	Fair match - some significant differences
D	Poor match - review recommended
X	Insufficient data - MAPE could not be calculated

Quality Threshold (is_good_match)

A well is considered a "good match" when ALL of these criteria are met:

* MAPE < 20%
* Correlation > 0.95
* Cumulative diff < 15%
* B-factor diff < 0.3

Ground Truth Report

The report (`ground_truth_report.txt`) includes:

```
PyForecast Ground Truth Comparison Report
==================================================

Summary:
  Total comparisons: 45
  Average MAPE: 12.3%
  Median MAPE: 10.5%
  Average correlation: 0.987
  Average cumulative diff: 8.2%
  Good match rate: 82.2%

Grade Distribution:
  A: 25 (55.6%)
  B: 12 (26.7%)
  C: 5 (11.1%)
  D: 3 (6.7%)

Detailed Results:
--------------------------------------------------

WELL-001/oil [A] GOOD
  ARIES:      qi=500.0, di=8.5%/yr, b=0.500
  pyforecast: qi=512.3, di=8.8%/yr, b=0.520
  Differences: qi=+2.5%, di=+3.5%, b=+0.020
  Metrics: MAPE=8.5%, corr=0.992, bias=+3.2%, cum_diff=+5.1%
```

Programmatic Usage

```
from pyforecast.import_ import AriesForecastImporter
from pyforecast.refinement import GroundTruthValidator, GroundTruthConfig

# Load ARIES forecasts
importer = AriesForecastImporter()
importer.load("aries_projections.csv")

# Configure comparison
config = GroundTruthConfig(comparison_months=60)
validator = GroundTruthValidator(importer, config)

# Compare a well
result = validator.validate(well, "oil")
```

```
if result:
    print(f"MAPE: {result.mape:.1f}%")
    print(f"Correlation: {result.correlation:.3f}")
    print(f"Good match: {result.is_good_match}")
    print(f"Grade: {result.match_grade}")
```

ARIES CSV File Format

The ARIES CSV file should have columns for well identifier and forecast expressions:

```
PROPNUM,OIL_EXPRESSION,GAS_EXPRESSION
42-001-00001,1000 X B/M 6 HYP B/0.50 8.5,5000 X M/M 6 EXP B/0.10 12
42-001-00002,500 X B/D 6 HYP B/0.75 10,
```

Supported unit codes:
* `B/M` - barrels/month (oil)
* `B/D` - barrels/day (oil)
* `M/M` - mcf/month (gas)
* `M/D` - mcf/day (gas)

Advanced Features

Rate Validation

Forecast rate arrays are automatically validated for problematic values:
* **NaN values** are replaced with 0 and logged as warnings
* **Infinite values** are clipped to 1e9 and logged as warnings
* **Negative values** are clipped to 0 and logged as warnings

This prevents silent calculation errors in metrics.

MAPE Edge Case Handling

When insufficient valid data points exist (fewer than 3 points above the 0.1 rate threshold), MAPE returns `None` instead of an incorrect value:
* `mape_valid` property indicates if MAPE was successfully calculated
* `is_good_match` returns `False` when MAPE is unavailable
* `match_grade` returns `"X"` for insufficient data

Identifier Mismatch Logging

The `validate_batch()` method tracks wells that exist in only one dataset:
* `wells_in_pyf_only`: Wells with pyforecast data but no ARIES data
* `wells_in_aries_only`: Wells with ARIES data but not in pyforecast batch

This helps diagnose ID normalization issues.

Parse Failure Logging

Unparseable ARIES expressions are logged with warnings and tracked:

```
importer = AriesForecastImporter()
importer.load("aries_projections.csv")

# Access parse failures
for well_id, product, expression in importer.parse_failures:
    print(f"Failed to parse {well_id}/{product}: '{expression}'")
```

Time-Series Export

Forecast arrays are exported to `ground_truth_timeseries.csv` for external visualization:

```
| Column | Description |
|--------|-------------|
| well_id | Well identifier |
| product | Product type (oil/gas/water) |
| month | Forecast month (0-based) |
| aries_rate | ARIES forecast rate |
| pyf_rate | pyforecast forecast rate |
| diff | Rate difference (pyf - aries) |
| pct_diff | Percentage difference |
```

Comparison Plots

Generate overlay plots with the `--gt-plots` flag:

```
pyforecast process data.csv --ground-truth aries.csv --gt-plots -o output/
```

Plots are saved to `output/ground_truth_plots/` with one PNG per well/product showing:
* ARIES curve (blue solid line)
* pyforecast curve (red dashed line)
* Metrics text box (MAPE, correlation, grade)

Lazy Loading

For large ARIES files (10,000+ wells), use lazy mode to reduce memory usage:

```
pyforecast process data.csv --ground-truth large_aries.csv --gt-lazy -o output/
```

In lazy mode:

* File is validated and rows counted during `load()`
* Data is streamed from disk on each `get()` call
* Uses constant memory regardless of file size
* Slower per-lookup but suitable for memory-constrained environments

Parallel Validation

For large batches, enable parallel validation:

```
pyforecast process data.csv --ground-truth aries.csv --gt-workers 4 -o output/
```

Uses `ThreadPoolExecutor` with configurable worker count. Default is 1 (sequential).

Regime Detection Calibration

Calibrate regime detection thresholds using known refrac/workover events.

Events File Format

```
well_id,event_date,event_type
42-001-00001,2022-06-15,refrac
42-001-00002,2023-01-20,workover
42-003-00005,2022-09-01,refrac
```

CLI Usage

```
pyforecast calibrate-regime data.csv --events known_events.csv -o calibration.json
```

Example Output

```
Loading production data from data.csv...
Loading known events from known_events.csv...
Found 45 known events

Testing regime detection thresholds...
  threshold=0.50: 92.0% detection rate (41/45)
  threshold=0.75: 88.0% detection rate (39/45)
  threshold=1.00: 78.0% detection rate (35/45)
  threshold=1.25: 65.0% detection rate (29/45)
  threshold=1.50: 52.0% detection rate (23/45)
  threshold=2.00: 35.0% detection rate (16/45)

Recommended threshold: 0.50 (92.0% detection rate)
```

Configuration Reference

Complete Refinement Configuration

```
refinement:
  # Logging settings
  enable_logging: false        # Log fit metadata to storage
  log_storage: sqlite          # Storage type: sqlite or csv
  log_path: null               # Path to storage (null = ~/.pyforecast/fit_logs.db)

  # Hindcast settings
  enable_hindcast: false       # Run hindcast validation
  hindcast_holdout_months: 6   # Months to hold out
  min_training_months: 12      # Minimum training data required

  # Residual analysis
  enable_residual_analysis: false

  # Regime calibration
  known_events_file: null      # CSV with known events

  # Parameter learning
  enable_learning: false

  # Ground truth comparison
  ground_truth_file: null      # ARIES AC_ECONOMIC CSV for comparison
  ground_truth_months: 60      # Months to compare forecasts
  ground_truth_lazy: false     # Stream file instead of loading into memory
  ground_truth_workers: 1      # Parallel workers (1 = sequential)
```

Ground Truth CLI Flags

Flag	Default	Description
`--ground-truth PATH`	-	ARIES AC_ECONOMIC CSV file for comparison
`--gt-months N`	60	Months to compare forecasts
`--gt-plots`	off	Generate comparison plots for each well
`--gt-lazy`	off	Stream ARIES file instead of loading into memory
`--gt-workers N`	1	Number of parallel workers for validation

Refinement Report Format

The refinement report (`refinement_report.txt`) is generated in the output directory when refinement features are enabled.

Sample Report

```
PyForecast Refinement Analysis Report
========================================

Hindcast Validation Summary:
  Wells with hindcast: 45
  Average MAPE: 17.3%
  Median MAPE: 14.8%
  Average correlation: 0.892
  Good hindcast rate: 82.2%

Hindcast Details:
----------------------------------------
  WELL-001/oil: MAPE=12.5%, corr=0.945, bias=-3.2% [GOOD]
  WELL-002/oil: MAPE=8.3%, corr=0.978, bias=1.5% [GOOD]
  WELL-003/oil: MAPE=35.2%, corr=0.654, bias=-18.5% [POOR]
  ...

Residual Analysis Summary:
  Total analyzed: 45
  With systematic patterns: 6 (13.3%)

Wells with systematic residual patterns:
----------------------------------------
  WELL-003/oil: DW=1.12, early_bias=-15.2%, late_bias=8.5%
  WELL-018/oil: DW=0.95, early_bias=12.8%, late_bias=-4.2%
```

Best Practices

Getting Started with Refinement

1. **Start with hindcast validation** - Measure baseline forecast accuracy
2. **Enable fit logging** - Accumulate data for learning
3. **Run residual analysis** - Identify systematic fit issues
4. **After 50+ fits** - Check parameter suggestions

Workflow for Continuous Improvement

```
# Initial processing with all refinement features
pyforecast process data.csv --hindcast --log-fits --residuals -o output/

# Review refinement report
cat output/refinement_report.txt

# After processing many wells, check suggestions
pyforecast suggest-params -p oil --basin "Permian"

# If suggestions look good, update config and re-run
pyforecast process new_data.csv -c updated_config.yaml -o output/
```

Interpreting Poor Hindcast Results

Symptom	Possible Cause	Action
High MAPE	Wrong decline behavior	Check b-factor bounds
Low correlation	Data noise or regime changes	Review regime detection
Large positive bias	Over-prediction	Reduce recency weighting
Large negative bias	Under-prediction	Increase recency weighting

When to Trust Parameter Suggestions

* [OK] High confidence (100+ samples)
* [OK] Basin/formation specific
* [OK] Good hindcast performance (MAPE < 20%)
* [!] Medium confidence - use with caution
* [X] Low confidence - gather more data first

API Reference

Data Classes

```
from pyforecast.refinement import (
    FitLogRecord,         # Complete fit metadata
    HindcastResult,       # Hindcast validation results
    ResidualDiagnostics,  # Residual analysis metrics
    GroundTruthResult,    # Ground truth comparison results
)
from pyforecast.refinement.schemas import ParameterSuggestion
```

Validators and Analyzers

```
from pyforecast.refinement import (
    HindcastValidator,       # Hindcast validation
    ResidualAnalyzer,        # Residual analysis
    FitLogger,               # Fit logging
    ParameterLearner,        # Parameter suggestions
    GroundTruthValidator,    # Ground truth comparison
    GroundTruthConfig,       # Ground truth configuration
    GroundTruthSummary,      # Batch validation summary with mismatch info
    summarize_ground_truth_results,  # Aggregate statistics
)
```

Plotting

```
from pyforecast.refinement.plotting import (
    plot_ground_truth_comparison,  # Single well plot
    plot_all_comparisons,          # Batch plot generation
)
```

ARIES Import

```
from pyforecast.import_ import (
    AriesForecastImporter,   # Load ARIES forecast files
    AriesForecastParams,     # Parsed forecast parameters
)
```

Storage

```
from pyforecast.refinement import FitLogStorage
from pyforecast.refinement.storage import get_default_storage_path

# Default path: ~/.pyforecast/fit_logs.db
storage = FitLogStorage()

# Custom path
storage = FitLogStorage("/path/to/custom.db")

# Query fit logs
records = storage.query(basin="Permian", product="oil", min_r_squared=0.8)

# Get statistics
stats = storage.get_statistics(basin="Permian")
```

Troubleshooting

"No fit logs found"

```
Error: No fit logs found at ~/.pyforecast/fit_logs.db
```

Solution: Run `pyforecast process --log-fits` first to create fit logs.

"Need at least 10 fits with hindcast data"

```
No parameter suggestions available.
Need at least 10 fits with hindcast data to generate suggestions.
```

Solution: Run `pyforecast process --log-fits --hindcast` on more wells.

High MAPE on all wells

Possible causes:

1. Data quality issues (check validation report)
2. Wrong b-factor bounds for your play
3. Regime detection not capturing correct decline period

Durbin-Watson consistently low

A DW < 1.5 suggests systematic underfitting. Try:

1. Adjusting b-factor bounds
2. Checking for missed regime changes
3. Reviewing recency weighting (may be too aggressive)