Pipeline-Sim Field Validation and Troubleshooting Guide

Field Validation Against Real Data

1. Data Collection Requirements

Minimum Required Measurements

- **Pressure**: At least at inlet/outlet of major segments
- **Flow Rate**: Total system flow and major branch flows
- **Temperature**: Process fluid temperature at key points
- **Fluid Properties**: Laboratory analysis of fluid samples

Recommended Additional Data

- **Elevation Profile**: Accurate pipe routing survey
- **Pipe Specifications**: As-built drawings with actual dimensions
- **Operating History**: Production logs over extended period
- **Equipment Performance**: Pump curves, valve Cv values

2. Validation Methodology

Step 1: Model Setup

python

```python
import pipeline_sim as ps
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, r2_score

# Load field data
field_data = pd.read_csv('field_measurements.csv', parse_dates=['timestamp'])

# Create network matching field configuration
network, fluid = ps.load_network('field_network.json')

# Verify network matches field
print(f"Network nodes: {len(network.nodes)}")
print(f"Network pipes: {len(network.pipes)}")
print(f"Data points: {len(field_data)}")
```

Step 2: Steady-State Validation

```
python
```

```python
def validate_steady_state(network, fluid, field_point):
    """Validate against single field measurement"""

    # Set boundary conditions from field data
    network.set_pressure(
        network.get_node('inlet'),
        field_point['inlet_pressure'] * 1e5  # Convert bar to Pa
    )
    network.set_flow_rate(
        network.get_node('outlet'),
        field_point['flow_rate'] / 3600  # Convert m³/h to m³/s
    )

    # Run simulation
    solver = ps.SteadyStateSolver(network, fluid)
    results = solver.solve()

    if not results.converged:
        return None

    # Compare results
    comparison = {
        'measured_outlet_pressure': field_point['outlet_pressure'],
        'simulated_outlet_pressure': results.node_pressures['outlet'] / 1e5,
        'measured_dp': field_point['inlet_pressure'] - field_point['outlet_pressure'],
        'simulated_dp': (results.node_pressures['inlet'] -
                         results.node_pressures['outlet']) / 1e5
    }

    return comparison

# Run validation for all data points
validations = []
```

```
for idx, row in field_data.iterrows():
    result = validate_steady_state(network, fluid, row)
    if result:
        validations.append(result)

# Calculate statistics
validation_df = pd.DataFrame(validations)
mae_pressure = mean_absolute_error(
    validation_df['measured_outlet_pressure'],
    validation_df['simulated_outlet_pressure']
)
r2_pressure = r2_score(
    validation_df['measured_outlet_pressure'],
    validation_df['simulated_outlet_pressure']
)

print(f"Mean Absolute Error: {mae_pressure:.2f} bar")
print(f"R² Score: {r2_pressure:.3f}")
```

Step 3: Correlation Tuning

```
python
```

```python
class TunedBeggsBrill(ps.FlowCorrelation):
    """Beggs-Brill with field-tuned parameters"""

    def __init__(self, tuning_factor=1.0):
        self.tuning_factor = tuning_factor
        self.base_correlation = ps.BeggsBrill()

    def calculate(self, fluid, pipe, flow_rate, p_in, t_in):
        # Get base calculation
        result = self.base_correlation.calculate(
            fluid, pipe, flow_rate, p_in, t_in
        )

        # Apply tuning factor
        result.pressure_gradient *= self.tuning_factor

        return result

# Optimize tuning factor
from scipy.optimize import minimize_scalar

def objective(tuning_factor):
    """Minimize prediction error"""
    correlation = TunedBeggsBrill(tuning_factor)
    errors = []

    for point in validation_data:
        # Run simulation with tuned correlation
        solver = ps.SteadyStateSolver(network, fluid)
        solver.set_correlation(correlation)
        results = solver.solve()

        if results.converged:
```

```
            error = abs(results.node_pressures['outlet'] / 1e5 -
                        point['outlet_pressure'])
            errors.append(error)

    return np.mean(errors)

# Find optimal tuning factor
result = minimize_scalar(objective, bounds=(0.8, 1.2))
optimal_tuning = result.x

print(f"Optimal tuning factor: {optimal_tuning:.3f}")
```

3. Common Validation Issues

Issue 1: Systematic Pressure Offset

Symptoms: Consistent over/under-prediction of pressure drop

Possible Causes:

- Incorrect fluid properties (density, viscosity)
- Wrong pipe roughness values
- Unaccounted pressure losses (fittings, valves)

Solution:

python

```python
# Adjust pipe roughness based on field data
def calibrate_roughness(network, field_data):
    """Calibrate pipe roughness from field measurements"""

    from scipy.optimize import least_squares

    def residuals(roughness_values):
        # Apply roughness values
        for pipe, roughness in zip(network.pipes.values(), roughness_values):
            pipe.roughness = roughness

        # Calculate errors
        errors = []
        for point in field_data:
            result = simulate_point(network, point)
            if result:
                errors.append(result['dp_error'])

        return errors

    # Initial guess (default roughness)
    x0 = [0.000045] * len(network.pipes)

    # Bounds: physical roughness range
    bounds = ([0.00001], [0.001])  # 0.01-1 mm

    # Optimize
    result = least_squares(residuals, x0, bounds=bounds)

    return result.x
```

Issue 2: Flow Pattern Mismatch

Symptoms: Good match at some conditions, poor at others

Possible Causes:

- Incorrect flow pattern prediction
- Phase fraction estimation errors
- Need for different correlation

Solution:

```
python
```

```python
# Test multiple correlations
correlations = [
    'Beggs-Brill',
    'Hagedorn-Brown',
    'Gray',
    'Mechanistic'
]

results_by_correlation = {}

for corr_name in correlations:
    errors = []

    for point in validation_data:
        solver = ps.SteadyStateSolver(network, fluid)
        solver.set_correlation(corr_name)
        result = solver.solve()

        if result.converged:
            error = calculate_error(result, point)
            errors.append(error)

    results_by_correlation[corr_name] = {
        'mae': np.mean(errors),
        'rmse': np.sqrt(np.mean(np.square(errors))),
        'max_error': np.max(errors)
    }

# Select best correlation
best_corr = min(results_by_correlation,
                key=lambda x: results_by_correlation[x]['mae'])
print(f"Best correlation: {best_corr}")
```

4. Troubleshooting Guide

Convergence Issues

Problem: Solver fails to converge

```
python
```

```python
# Diagnostic function
def diagnose_convergence(network, fluid):
    """Diagnose convergence issues"""

    solver = ps.SteadyStateSolver(network, fluid)
    solver.config.verbose = True
    solver.config.max_iterations = 50  # Limit for diagnostics

    results = solver.solve()

    if not results.converged:
        print("Convergence Diagnostics:")
        print(f"Final residual: {results.residual:.2e}")
        print(f"Iterations: {results.iterations}")

        # Check for common issues

        # 1. Disconnected nodes
        disconnected = []
        for node_id, node in network.nodes.items():
            upstream = network.get_upstream_pipes(node)
            downstream = network.get_downstream_pipes(node)
            if not upstream and not downstream:
                disconnected.append(node_id)

        if disconnected:
            print(f"Disconnected nodes: {disconnected}")

        # 2. Over-constrained system
        num_pressure_bcs = len(network.pressure_specs)
        num_flow_bcs = len(network.flow_specs)
        num_nodes = len(network.nodes)
```

```
        print(f"Pressure BCs: {num_pressure_bcs}")
        print(f"Flow BCs: {num_flow_bcs}")
        print(f"Degrees of freedom: {num_nodes - num_pressure_bcs}")

        # 3. Bad initial guess
        print("\nTrying with different initial guess...")
        solver.set_initial_pressure(50e5)  # 50 bar
        results2 = solver.solve()

        if results2.converged:
            print("Converged with different initial guess!")

        # 4. Relaxation factor
        print("\nTrying with lower relaxation factor...")
        solver.config.relaxation_factor = 0.3
        results3 = solver.solve()

        if results3.converged:
            print("Converged with lower relaxation!")

    return results
```

Solution Strategies:

1. **Improve Initial Guess**

python

```python
# Use previous solution as initial guess
previous_results = None

for time_step in simulation_steps:
    solver = ps.SteadyStateSolver(network, fluid)

    if previous_results:
        solver.set_initial_guess(previous_results)

    results = solver.solve()
    previous_results = results
```

2. **Ramping Strategy**

python

```python
def solve_with_ramping(network, fluid, target_flow):
    """Gradually increase flow to target"""

    ramp_steps = 10
    flows = np.linspace(0.01, target_flow, ramp_steps)

    results = None
    for flow in flows:
        network.set_flow_rate(network.get_node('outlet'), flow)

        solver = ps.SteadyStateSolver(network, fluid)
        if results:
            solver.set_initial_guess(results)

        results = solver.solve()

        if not results.converged:
            print(f"Failed at flow = {flow:.3f} m³/s")
            break

    return results
```

Performance Issues

Problem: Simulation too slow for large networks

Solutions:

1. **Enable Parallel Computing**

python

```python
solver.config.use_parallel = True
solver.config.num_threads = 8  # Or os.cpu_count()
```

2. **Simplify Network**

python

```python
def simplify_network(network, merge_threshold=50):
    """Merge short pipe segments"""

    simplified = ps.Network()

    # Copy nodes
    for node_id, node in network.nodes.items():
        simplified.add_node(node_id, node.type)

    # Merge adjacent pipes
    merged_pipes = []
    for pipe in network.pipes.values():
        if pipe.length < merge_threshold:
            # Find adjacent pipe to merge with
            # ... merging logic ...
            pass
        else:
            simplified.add_pipe(
                pipe.id,
                pipe.upstream,
                pipe.downstream,
                pipe.length,
                pipe.diameter
            )

    return simplified
```

3. **Use Adaptive Tolerance**

python

```python
# Start with loose tolerance, then refine
tolerances = [1e-3, 1e-4, 1e-5, 1e-6]

for tol in tolerances:
    solver.config.tolerance = tol
    results = solver.solve()

    if not results.converged:
        print(f"Failed at tolerance = {tol}")
        break
```

5. Validation Report Template

```
python
```

```python
def generate_validation_report(network, field_data, simulation_results):
    """Generate comprehensive validation report"""

    report = {
        'summary': {
            'date': datetime.now().isoformat(),
            'network_name': 'Field Pipeline Network',
            'data_points': len(field_data),
            'simulation_points': len(simulation_results)
        },
        'statistics': {
            'pressure': {
                'mae': calculate_mae(field_data, simulation_results, 'pressure'),
                'rmse': calculate_rmse(field_data, simulation_results, 'pressure'),
                'r2': calculate_r2(field_data, simulation_results, 'pressure'),
                'max_error': calculate_max_error(field_data, simulation_results, 'pressur
            },
            'flow': {
                'mae': calculate_mae(field_data, simulation_results, 'flow'),
                'rmse': calculate_rmse(field_data, simulation_results, 'flow'),
                'r2': calculate_r2(field_data, simulation_results, 'flow')
            }
        },
        'calibration': {
            'roughness_values': get_calibrated_roughness(network),
            'tuning_factors': get_tuning_factors(),
            'selected_correlation': get_best_correlation()
        }
    }

    # Generate plots
    fig, axes = plt.subplots(2, 2, figsize=(12, 10))
```

```
# Parity plot - Pressure
ax = axes[0, 0]
ax.scatter(field_data['pressure'], simulation_results['pressure'], alpha=0.6)
ax.plot([0, 100], [0, 100], 'r--', label='Perfect match')
ax.set_xlabel('Measured Pressure (bar)')
ax.set_ylabel('Simulated Pressure (bar)')
ax.set_title('Pressure Validation')
ax.legend()

# Error distribution
ax = axes[0, 1]
errors = simulation_results['pressure'] - field_data['pressure']
ax.hist(errors, bins=30, edgecolor='black')
ax.set_xlabel('Pressure Error (bar)')
ax.set_ylabel('Frequency')
ax.set_title('Error Distribution')

# Time series comparison
ax = axes[1, 0]
ax.plot(field_data['timestamp'], field_data['pressure'],
        'b-', label='Measured', alpha=0.7)
ax.plot(field_data['timestamp'], simulation_results['pressure'],
        'r--', label='Simulated', alpha=0.7)
ax.set_xlabel('Time')
ax.set_ylabel('Pressure (bar)')
ax.set_title('Time Series Comparison')
ax.legend()

# Flow validation
ax = axes[1, 1]
ax.scatter(field_data['flow'], simulation_results['flow'], alpha=0.6)
ax.plot([0, max(field_data['flow'])], [0, max(field_data['flow'])],
        'r--', label='Perfect match')
```

```
    ax.set_xlabel('Measured Flow (m³/h)')
    ax.set_ylabel('Simulated Flow (m³/h)')
    ax.set_title('Flow Validation')
    ax.legend()

    plt.tight_layout()
    plt.savefig('validation_report.png', dpi=300)

    # Save report
    with open('validation_report.json', 'w') as f:
        json.dump(report, f, indent=2)

    return report
```

6. Best Practices for Field Validation

Data Quality Checks

1. **Sensor Calibration**: Verify all sensors are recently calibrated
2. **Steady-State Conditions**: Ensure measurements during stable operation
3. **Mass Balance**: Check field data satisfies conservation laws
4. **Outlier Detection**: Remove or investigate anomalous readings

Model Refinement Process

1. Start with nominal values from design
2. Validate against steady-state data first
3. Tune parameters systematically (roughness → fluid properties → correlations)
4. Validate against transient events
5. Document all assumptions and modifications

Continuous Improvement

```
python
```

```python
class ModelUpdateManager:
    """Manage model updates based on new field data"""

    def __init__(self, base_model_file):
        self.base_model = base_model_file
        self.update_history = []

    def update_model(self, new_field_data, update_type='auto'):
        """Update model with new field data"""

        # Load current model
        network, fluid = ps.load_network(self.base_model)

        # Run validation
        validation_results = validate_against_field(network, fluid, new_field_data)

        if update_type == 'auto':
            if validation_results['mae'] > 2.0:  # 2 bar threshold
                # Significant error - recalibrate
                self.recalibrate_model(network, fluid, new_field_data)
            elif validation_results['mae'] > 1.0:
                # Minor adjustment needed
                self.fine_tune_model(network, fluid, new_field_data)

        # Log update
        self.update_history.append({
            'timestamp': datetime.now(),
            'data_points': len(new_field_data),
            'mae_before': validation_results['mae'],
            'update_type': update_type
        })

    def generate_trending_report(self):
```

```
    """Track model performance over time"""

    df = pd.DataFrame(self.update_history)

    fig, ax = plt.subplots(figsize=(10, 6))
    ax.plot(df['timestamp'], df['mae_before'], 'o-')
    ax.set_xlabel('Date')
    ax.set_ylabel('Mean Absolute Error (bar)')
    ax.set_title('Model Performance Trend')
    ax.grid(True, alpha=0.3)

    # Add update markers
    for idx, row in df.iterrows():
        if row['update_type'] == 'recalibrate':
            ax.axvline(x=row['timestamp'], color='red',
                       linestyle='--', alpha=0.5)

    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig('model_performance_trend.png')
```

Quick Troubleshooting Reference

Common Error Messages

Error	Cause	Solution
"Matrix decomposition failed"	Singular system matrix	Check boundary conditions
"Maximum iterations reached"	Poor convergence	Reduce relaxation factor
"Negative pressure detected"	Unrealistic flow/pressure	Check units and BCs
"Memory allocation failed"	Large network	Enable sparse matrix mode
"Invalid correlation"	Missing plugin	Install correlation package

Performance Optimization Checklist

- [ ] Enable parallel computing for networks > 1000 nodes
- [ ] Use sparse matrix storage for networks > 5000 nodes
- [ ] Simplify network by merging short segments
- [ ] Start with coarse tolerance, refine if needed
- [ ] Use previous solution as initial guess
- [ ] Profile code to identify bottlenecks
- [ ] Consider cloud deployment for very large networks

Emergency Contacts

- GitHub Issues: https://github.com/pipeline-sim/pipeline-sim/issues
- Community Forum: https://forum.pipeline-sim.org
- Commercial Support: support@pipeline-sim.com

Remember: Good validation requires good data. Invest in quality measurements for reliable simulations.