PROFESSIONAL GUIDE: Pipeline-Sim Model Construction

Table of Contents

Quick Reference: Available Methods

Node Methods

```python
# Properties (read-only)
node.id                 # Node identifier
node.type               # NodeType enum
node.pressure           # Current pressure (Pa)
node.temperature        # Current temperature (K)
node.elevation          # Elevation (m) - READ ONLY!

# Boundary Conditions
node.set_pressure_bc(pressure_pa)    # Set pressure BC (MUST USE for sources/sinks!)
node.has_pressure_bc()               # Check if BC is set
node.pressure_bc()                   # Get BC value

# Flow Control
node.set_fixed_flow_rate(flow_m3s)   # Set flow rate (m³/s)
node.set_flow_rate(flow_m3s)         # Alias for above
node.fixed_flow_rate()               # Get flow rate

# Equipment
node.set_pump_speed(speed)           # 0-1 normalized
node.set_pump_curve(a, b)            # Head = a - b*Q²
node.set_compressor_ratio(ratio)     # Pressure ratio
```

Pipe Methods

python

```
# Properties (read-only)
pipe.id                 # Pipe identifier
pipe.length             # Length (m)
pipe.diameter           # Diameter (m)
pipe.upstream           # Upstream node
pipe.downstream         # Downstream node
pipe.area()             # Cross-sectional area (m²)
pipe.volume()           # Pipe volume (m³)

# Configurable Properties
pipe.set_roughness(epsilon)       # Absolute roughness (m)
pipe.set_inclination(angle_rad)   # Inclination angle (radians)

# Results (after solving)
pipe.flow_rate()                  # Flow rate (m³/s)
pipe.velocity()                   # Velocity (m/s)
pipe.reynolds_number(mu, rho)     # Reynolds number
pipe.friction_factor(Re)          # Friction factor
```

Network Methods

python

```python
# Building
network.add_node(id, node_type)      # Returns Node object
network.add_pipe(id, upstream, downstream, length, diameter)  # Returns Pipe object

# Access
network.get_node(id)              # Get node by ID
network.get_pipe(id)              # Get pipe by ID
network.nodes()                   # All nodes dict
network.pipes()                   # All pipes dict

# Boundary Conditions (DON'T USE - sets pressure but not BC!)
network.set_pressure(node, pressure) # WRONG - use node.set_pressure_bc()
network.set_flow_rate(node, flow)    # Sets flow demand

# Queries
network.node_count()
network.pipe_count()
network.is_valid()
network.get_upstream_pipes(node)
network.get_downstream_pipes(node)

# I/O
network.load_from_json(filename)
network.save_to_json(filename)
```

Basic Pipeline Models

1. Simple Pipeline (Two Nodes)

python

```python
import pipeline_sim as ps

# Create network
network = ps.Network()

# Create nodes
inlet = network.add_node("INLET", ps.NodeType.SOURCE)
outlet = network.add_node("OUTLET", ps.NodeType.SINK)

# Create pipe
pipe = network.add_pipe("PIPE-1", inlet, outlet,
                        length=1000.0,    # meters
                        diameter=0.3048)  # meters (12")

# Configure pipe
pipe.set_roughness(0.000045)  # Commercial steel

# SET BOUNDARY CONDITIONS (CRITICAL!)
inlet.set_pressure_bc(70e5)   # 70 bar
outlet.set_pressure_bc(65e5)  # 65 bar

# Create fluid
fluid = ps.FluidProperties()
fluid.oil_density = 850.0      # kg/m³
fluid.oil_viscosity = 0.002    # Pa.s
fluid.oil_fraction = 1.0
fluid.gas_fraction = 0.0
fluid.water_fraction = 0.0

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

print(f"Flow rate: {results.pipe_flow_rates['PIPE-1']} m³/s")
```

2. Three-Node Network (Branching)

python

```python
# Create network
network = ps.Network()

# Nodes
source = network.add_node("SOURCE", ps.NodeType.SOURCE)
junction = network.add_node("JUNCTION", ps.NodeType.JUNCTION)
sink1 = network.add_node("SINK1", ps.NodeType.SINK)
sink2 = network.add_node("SINK2", ps.NodeType.SINK)

# Pipes
supply = network.add_pipe("SUPPLY", source, junction, 1000, 0.4)
branch1 = network.add_pipe("BRANCH1", junction, sink1, 500, 0.3)
branch2 = network.add_pipe("BRANCH2", junction, sink2, 500, 0.3)

# Configure
for pipe in [supply, branch1, branch2]:
    pipe.set_roughness(0.000045)

# Boundary conditions
source.set_pressure_bc(100e5)   # 100 bar
sink1.set_pressure_bc(80e5)     # 80 bar
sink2.set_pressure_bc(85e5)     # 85 bar

# Note: Junction pressure will be calculated!
```

3. Vertical Pipeline (With Elevation)

python

```python
# Create vertical riser
network = ps.Network()

bottom = network.add_node("BOTTOM", ps.NodeType.SOURCE)
top = network.add_node("TOP", ps.NodeType.SINK)

# Note: Can't set elevation directly! Work with pressure BCs
# If you need elevation effects, adjust pressures accordingly
# Hydrostatic pressure = ρ * g * h

riser = network.add_pipe("RISER", bottom, top, 100, 0.2)
riser.set_roughness(0.000045)
riser.set_inclination(1.5708)  # 90 degrees in radians

# Account for 100m elevation in pressure BCs
# Assume water: ΔP_hydrostatic = 1000 * 9.81 * 100 = 9.81 bar
bottom.set_pressure_bc(30e5)    # 30 bar
top.set_pressure_bc(10e5)       # 10 bar (20 bar driving - 9.81 hydrostatic)
```

Complex Network Models

4. Production Manifold System

python

```python
def create_production_network():
    network = ps.Network()

    # Wells (different pressures)
    wells = []
    well_data = [
        ("WELL-1", 320e5),  # 320 bar
        ("WELL-2", 310e5),  # 310 bar
        ("WELL-3", 300e5),  # 300 bar
    ]

    for name, pressure in well_data:
        well = network.add_node(name, ps.NodeType.SOURCE)
        well.set_pressure_bc(pressure)
        wells.append(well)

    # Manifold
    manifold = network.add_node("MANIFOLD", ps.NodeType.JUNCTION)

    # Separator
    separator = network.add_node("SEPARATOR", ps.NodeType.SINK)
    separator.set_pressure_bc(50e5)  # 50 bar

    # Flowlines
    for i, well in enumerate(wells):
        pipe = network.add_pipe(f"FLOWLINE-{i+1}", well, manifold,
                                length=2000 + i*500,  # Different lengths
                                diameter=0.2032)      # 8"
        pipe.set_roughness(0.000045)

    # Trunk line
    trunk = network.add_pipe("TRUNK", manifold, separator, 5000, 0.4064)  # 16"
    trunk.set_roughness(0.000045)

    return network
```

5. Looped Network

```python
def create_looped_network():
    network = ps.Network()

    # Create a simple loop
    nodes = {}
    for i in range(4):
        nodes[i] = network.add_node(f"NODE-{i}", ps.NodeType.JUNCTION)

    # Source and sink
    source = network.add_node("SOURCE", ps.NodeType.SOURCE)
    sink = network.add_node("SINK", ps.NodeType.SINK)

    # Connect in a loop
    network.add_pipe("P01", nodes[0], nodes[1], 1000, 0.3)
    network.add_pipe("P12", nodes[1], nodes[2], 1000, 0.3)
    network.add_pipe("P23", nodes[2], nodes[3], 1000, 0.3)
    network.add_pipe("P30", nodes[3], nodes[0], 1000, 0.3)

    # Connect source and sink
    network.add_pipe("P_IN", source, nodes[0], 500, 0.4)
    network.add_pipe("P_OUT", nodes[2], sink, 500, 0.4)

    # Set all roughness
    for pipe in network.pipes().values():
        pipe.set_roughness(0.000045)

    # Boundary conditions
    source.set_pressure_bc(100e5)
    sink.set_pressure_bc(90e5)

    return network
```

Equipment Models

6. Pipeline with Pump

python

```python
def create_pumped_pipeline():
    network = ps.Network()

    # Nodes
    suction = network.add_node("SUCTION", ps.NodeType.SOURCE)
    pump = network.add_node("PUMP", ps.NodeType.PUMP)
    discharge = network.add_node("DISCHARGE", ps.NodeType.SINK)

    # Pipes
    suction_pipe = network.add_pipe("SUCTION_PIPE", suction, pump, 100, 0.4)
    discharge_pipe = network.add_pipe("DISCHARGE_PIPE", pump, discharge, 5000, 0.3)

    # Configure
    suction_pipe.set_roughness(0.000045)
    discharge_pipe.set_roughness(0.000045)

    # Pump configuration
    pump.set_pump_speed(1.0)      # 100% speed
    pump.set_pump_curve(150, 0.01) # Head = 150 - 0.01*Q² (meters)

    # Boundary conditions
    suction.set_pressure_bc(2e5)    # 2 bar (low pressure)
    discharge.set_pressure_bc(50e5) # 50 bar (high pressure)

    return network
```

7. Gas Pipeline with Compressor

python

```python
def create_gas_pipeline():
    network = ps.Network()

    # Nodes
    inlet = network.add_node("INLET", ps.NodeType.SOURCE)
    compressor = network.add_node("COMPRESSOR", ps.NodeType.COMPRESSOR)
    outlet = network.add_node("OUTLET", ps.NodeType.SINK)

    # Pipes
    inlet_pipe = network.add_pipe("INLET_PIPE", inlet, compressor, 1000, 0.6)
    outlet_pipe = network.add_pipe("OUTLET_PIPE", compressor, outlet, 50000, 0.6)

    # Configure
    inlet_pipe.set_roughness(0.00002)  # Smooth pipe for gas
    outlet_pipe.set_roughness(0.00002)

    # Compressor
    compressor.set_compressor_ratio(2.5)  # Compress by factor of 2.5

    # Boundary conditions
    inlet.set_pressure_bc(30e5)   # 30 bar inlet
    outlet.set_pressure_bc(70e5)  # 70 bar outlet

    # Gas properties
    fluid = ps.FluidProperties()
    fluid.gas_fraction = 1.0
    fluid.oil_fraction = 0.0
    fluid.water_fraction = 0.0
    fluid.gas_density = 25.0     # kg/m³ at operating conditions
    fluid.gas_viscosity = 0.00001 # Pa.s

    return network, fluid
```

Multiphase Flow Models

8. Oil-Water Flow

python

```python
def create_multiphase_pipeline():
    network = ps.Network()

    inlet = network.add_node("INLET", ps.NodeType.SOURCE)
    outlet = network.add_node("OUTLET", ps.NodeType.SINK)

    pipe = network.add_pipe("MULTIPHASE", inlet, outlet, 10000, 0.3048)
    pipe.set_roughness(0.000045)

    # Boundary conditions
    inlet.set_pressure_bc(100e5)  # 100 bar
    outlet.set_pressure_bc(20e5)  # 20 bar

    # Multiphase fluid
    fluid = ps.FluidProperties()

    # Phase fractions
    fluid.oil_fraction = 0.7
    fluid.water_fraction = 0.3
    fluid.gas_fraction = 0.0

    # Phase properties
    fluid.oil_density = 850.0
    fluid.water_density = 1025.0
    fluid.oil_viscosity = 0.005
    fluid.water_viscosity = 0.001

    # Additional properties
    fluid.water_cut = 0.3  # 30% water cut

    print(f"Mixture density: {fluid.mixture_density()} kg/m³")
    print(f"Mixture viscosity: {fluid.mixture_viscosity()*1000} cP")

    return network, fluid
```

9. Three-Phase Flow (Oil-Water-Gas)

python

```python
def create_three_phase_flow():
    fluid = ps.FluidProperties()

    # Phase fractions (must sum to 1.0)
    fluid.oil_fraction = 0.6
    fluid.water_fraction = 0.3
    fluid.gas_fraction = 0.1

    # Densities at operating conditions
    fluid.oil_density = 800.0    # kg/m³
    fluid.water_density = 1025.0 # kg/m³
    fluid.gas_density = 50.0     # kg/m³ (at pressure)

    # Viscosities
    fluid.oil_viscosity = 0.003    # Pa.s
    fluid.water_viscosity = 0.001  # Pa.s
    fluid.gas_viscosity = 0.00002  # Pa.s

    # PVT properties
    fluid.gas_oil_ratio = 150.0    # sm³/sm³
    fluid.water_cut = 0.33         # Water/(Oil+Water)

    # Operating conditions
    fluid.temperature = 60 + 273.15  # 60°C
    fluid.pressure = 50e5            # 50 bar

    return fluid
```

Solver Configuration

10. Basic Solver Setup

```python
solver = ps.SteadyStateSolver(network, fluid)

# Access configuration
config = solver.config

# Basic settings
config.tolerance = 1e-6          # Convergence tolerance
config.max_iterations = 100      # Maximum iterations
config.verbose = True            # Print progress

# Relaxation
config.relaxation_factor = 1.0   # 1.0 = no relaxation
config.use_adaptive_relaxation = True
config.min_relaxation = 0.1
config.max_relaxation = 1.0

# Line search
config.use_line_search = True
config.line_search_alpha = 1e-4  # Armijo constant
config.line_search_beta = 0.5    # Backtracking factor

# Solve
results = solver.solve()
```

11. Advanced Solver Configuration

```python
# For difficult problems
config = solver.config

# Jacobian method
config.jacobian_method = ps.JacobianMethod.ANALYTICAL  # or FINITE_DIFFERENCE
config.finite_diff_step = 1000.0  # Pa (for finite diff)

# Linear solver
config.linear_solver = ps.LinearSolver.LU_DECOMPOSITION  # Default
# Other options: QR_DECOMPOSITION, ITERATIVE_BICGSTAB

# Trust region (for highly nonlinear problems)
config.use_trust_region = True
config.trust_region_radius = 1e6  # Pa

# Convergence criteria
config.check_relative_tolerance = True
config.relative_tolerance = 1e-8
config.stagnation_check_window = 5
config.stagnation_tolerance = 1e-10
```

Common Patterns and Best Practices

Pattern 1: Building Large Networks

python

```python
def build_field_network(num_wells, num_platforms):
    network = ps.Network()

    # Store nodes for easy access
    wells = {}
    platforms = {}

    # Create wells
    for i in range(num_wells):
        well = network.add_node(f"WELL-{i+1}", ps.NodeType.SOURCE)
        well.set_pressure_bc((300 + i*10) * 1e5)  # Varying pressures
        wells[i] = well

    # Create platforms
    for j in range(num_platforms):
        platform = network.add_node(f"PLATFORM-{j+1}", ps.NodeType.JUNCTION)
        platforms[j] = platform

    # Connect wells to nearest platform
    for i, well in wells.items():
        platform_id = i % num_platforms
        pipe = network.add_pipe(
            f"FLOW-{i+1}",
            well,
            platforms[platform_id],
            length=2000 + np.random.rand()*1000,
            diameter=0.2032
        )
        pipe.set_roughness(0.000045)

    return network, wells, platforms
```

Pattern 2: Parameter Sweep

python

```python
def parameter_sweep(network, fluid, pressures):
    """Test network performance at different outlet pressures"""

    results_data = []
    outlet = network.get_node("OUTLET")

    for pressure in pressures:
        # Update boundary condition
        outlet.set_pressure_bc(pressure)

        # Solve
        solver = ps.SteadyStateSolver(network, fluid)
        solver.config.verbose = False
        results = solver.solve()

        if results.converged:
            total_flow = sum(abs(q) for q in results.pipe_flow_rates.values())
            results_data.append({
                'pressure': pressure/1e5,
                'total_flow': total_flow,
                'iterations': results.iterations
            })

    return results_data
```

Pattern 3: Error Handling

python

```python
def robust_solve(network, fluid, max_attempts=3):
    """Solve with automatic parameter adjustment on failure"""

    solver = ps.SteadyStateSolver(network, fluid)

    for attempt in range(max_attempts):
        if attempt == 0:
            # First attempt - default settings
            solver.config.relaxation_factor = 1.0
        elif attempt == 1:
            # Second attempt - with relaxation
            solver.config.relaxation_factor = 0.7
            solver.config.use_line_search = True
        else:
            # Final attempt - conservative settings
            solver.config.relaxation_factor = 0.5
            solver.config.max_iterations = 200
            solver.config.use_adaptive_relaxation = True

        results = solver.solve()

        if results.converged:
            print(f"Converged on attempt {attempt + 1}")
            return results

    print("Failed to converge after all attempts")
    return results
```

Troubleshooting

Common Issues and Solutions

1. **"Network has no pressure boundary conditions"**

   python

   ```python
   # WRONG:
   network.set_pressure(node, pressure)  # Doesn't set BC!

   # CORRECT:
   node.set_pressure_bc(pressure)       # Sets BC properly
   ```

2. **Solver doesn't converge**

```python
# Try these in order:
solver.config.relaxation_factor = 0.7
solver.config.use_line_search = True
solver.config.use_adaptive_relaxation = True
solver.config.max_iterations = 200
```

3. **Can't set elevation**

```python
# Elevation is read-only in current bindings
# Workaround: Include hydrostatic pressure in BCs

dP_hydrostatic = rho * g * h  # Pa
bottom_pressure = top_pressure + dP_hydrostatic
```

4. **Getting bound method instead of value**

```python
# Some properties might not work correctly
# Try using the method directly:
value = pipe.roughness()  # If pipe.roughness doesn't work
```

Validation Checklist

- [ ] All source/sink nodes have pressure BCs set with `set_pressure_bc()`
- [ ] All pipes have roughness set
- [ ] Fluid properties sum to 1.0 for phase fractions
- [ ] Network has at least one pressure BC
- [ ] No isolated sections in network
- [ ] Reasonable initial guesses for iterative solution

Example: Complete Field Development

python

```
def create_complete_field():
    """Professional example combining all concepts"""

    network = ps.Network()

    # 1. Create subsea wells
    print("Creating subsea wells...")
    wells = []
    for i, (name, pressure) in enumerate([
        ("WELL-A1", 350e5), ("WELL-A2", 340e5),
        ("WELL-B1", 360e5), ("WELL-B2", 355e5)
    ]):
        well = network.add_node(name, ps.NodeType.SOURCE)
        well.set_pressure_bc(pressure)
        wells.append(well)

    # 2. Subsea manifolds
    print("Creating manifolds...")
    manifold_a = network.add_node("MANIFOLD-A", ps.NodeType.JUNCTION)
    manifold_b = network.add_node("MANIFOLD-B", ps.NodeType.JUNCTION)

    # 3. Platform facilities
    print("Creating platform...")
    platform = network.add_node("PLATFORM", ps.NodeType.JUNCTION)
    hp_sep = network.add_node("HP-SEP", ps.NodeType.JUNCTION)
    pump_node = network.add_node("EXPORT-PUMP", ps.NodeType.PUMP)
    export = network.add_node("EXPORT", ps.NodeType.SINK)

    # 4. Connect with pipes
    print("Creating pipeline network...")

    # Flowlines
    for i in range(2):
        pipe = network.add_pipe(f"FLOW-A{i+1}", wells[i], manifold_a,
                                2500, 0.2032)
        pipe.set_roughness(0.000045)

    for i in range(2):
        pipe = network.add_pipe(f"FLOW-B{i+1}", wells[i+2], manifold_b,
                                3000, 0.2032)
        pipe.set_roughness(0.000045)

    # Trunk lines
```

```
trunk_a = network.add_pipe("TRUNK-A", manifold_a, platform, 5000, 0.4064)
trunk_b = network.add_pipe("TRUNK-B", manifold_b, platform, 4000, 0.4064)
trunk_a.set_roughness(0.000045)
trunk_b.set_roughness(0.000045)

# Platform piping
sep_line = network.add_pipe("TO-SEP", platform, hp_sep, 100, 0.6096)
pump_suct = network.add_pipe("PUMP-SUCT", hp_sep, pump_node, 50, 0.5080)
pump_disch = network.add_pipe("PUMP-DISCH", pump_node, export, 200, 0.4064)

for pipe in [sep_line, pump_suct, pump_disch]:
    pipe.set_roughness(0.000030)  # Smooth platform piping

# 5. Configure pump
pump_node.set_pump_speed(1.0)
pump_node.set_pump_curve(200, 0.005)  # Boost pressure

# 6. Set export pressure
export.set_pressure_bc(150e5)  # 150 bar export

# 7. Create realistic fluid
fluid = ps.FluidProperties()
fluid.oil_fraction = 0.75
fluid.water_fraction = 0.20
fluid.gas_fraction = 0.05
fluid.oil_density = 820.0
fluid.water_density = 1025.0
fluid.gas_density = 80.0  # At separator conditions
fluid.oil_viscosity = 0.002
fluid.water_viscosity = 0.001
fluid.gas_viscosity = 0.00002

# 8. Solve with appropriate settings
solver = ps.SteadyStateSolver(network, fluid)
solver.config.verbose = True
solver.config.tolerance = 1e-6
solver.config.use_line_search = True
solver.config.use_adaptive_relaxation = True

print("\nSolving network...")
results = solver.solve()

if results.converged:
    print(f"\nSuccess! Converged in {results.iterations} iterations")
```

```
    # Calculate total production
    total = sum(abs(results.pipe_flow_rates[f"FLOW-{w}"])
                for w in ["A1", "A2", "B1", "B2"])
    print(f"Total production: {total*86400:.0f} m³/day")

  return network, fluid, results

# Run the example
if __name__ == "__main__":
  network, fluid, results = create_complete_field()
```

This guide covers all the essential patterns for using Pipeline-Sim effectively. Remember the key points:

1. Always use `node.set_pressure_bc()` for boundary conditions
2. Configure all pipe properties (especially roughness)
3. Ensure fluid fractions sum to 1.0
4. Use appropriate solver settings for your problem type
5. Handle non-convergence with relaxation and line search