GAT: A High-Performance Rust Toolkit for Power System Analysis

Tom Wilson
`https://github.com/monistowl/gat`

November 2024 — Draft 6 (Expanded)

Abstract

We present the Grid Analysis Toolkit (GAT), an open-source command-line toolkit for power system analysis implemented in Rust. This comprehensive technical reference documents GAT's complete solver hierarchy for optimal power flow (OPF)—from sub-millisecond economic dispatch through DC-OPF, SOCP relaxation, and full nonlinear AC-OPF with IPOPT—alongside state estimation, N-k contingency analysis, and time-series dispatch. We detail the framework's design decisions rooted in Rust's type system and memory safety guarantees, the challenges of parsing heterogeneous power system datasets (MATPOWER, PSS/E, CIM, pandapower), and the mathematical foundations underlying each analysis module. Extensive benchmarks against PGLib-OPF demonstrate convergence to reference objective values within 0.01% for standard IEEE test cases. This paper targets doctoral-level power systems engineers and provides complete mathematical formulations, algorithmic pseudocode, implementation insights, and numerical considerations for reproducibility.

Contents

Part I

Framework Architecture

1 Introduction

The Optimal Power Flow (OPF) problem is fundamental to power system operations, determining the economically optimal generator dispatch subject to physical network constraints. First formulated by Carpentier in 1962 [1], OPF remains computationally challenging due to the non-convex nature of AC power flow equations. Modern grid operations require not only OPF solutions but also state estimation from SCADA measurements, contingency analysis for reliability assessment, and time-series analysis for renewable integration studies.

1.1 Motivation and Design Goals

Existing power system analysis tools present significant barriers to adoption:

1. **Proprietary licensing**: Commercial tools (PowerWorld, PSS/E, PSCAD) require expensive licenses
2. **Runtime dependencies**: MATPOWER requires MATLAB; PowerModels.jl requires Julia's package ecosystem
3. **Installation complexity**: IPOPT, HSL solvers, and SuiteSparse require careful configuration
4. **Language fragmentation**: Python (pandapower, PyPSA), Julia (PowerModels), MATLAB (MATPOWER) create interoperability challenges
5. **Performance limitations**: Interpreted languages incur overhead; GC pauses affect real-time applications

GAT addresses these limitations through five design principles:

Single-binary deployment Self-contained executable with no runtime dependencies beyond libc

Memory safety without GC Rust's ownership system prevents buffer overflows, use-after-free, and data races at compile time

Type-driven correctness Newtype wrappers distinguish bus IDs from generator IDs; units are encoded in types

Composable data pipelines Apache Arrow/Parquet output integrates with Python, R, DuckDB, and Spark

Modular solver backends LP (HiGHS, CBC), conic (Clarabel), and NLP (IPOPT, L-BFGS) solvers are interchangeable

1.2 Contributions

This paper makes the following contributions:

1. A comprehensive open-source power system analysis toolkit in Rust covering OPF, state estimation, and contingency analysis

2. Type-safe data modeling using Rust's algebraic data types and newtype patterns
3. Analytical Jacobian and Hessian derivations for IPOPT-backed AC-OPF with full thermal constraints
4. Dataset interoperability layer handling MATPOWER, PSS/E RAW, CIM XML, and pandapower JSON formats
5. PTDF/LODF-based fast contingency screening for N-k analysis
6. Validation against PGLib-OPF, OPFData, and PFΔ benchmark suites
7. Detailed numerical considerations for floating-point stability in power system computations

2 Framework Design Decisions

2.1 Why Rust?

The choice of Rust as the implementation language reflects several technical requirements:

2.1.1 Memory Safety Without Garbage Collection

Power system analysis involves large sparse matrices (Y-bus for 10,000+ bus systems) and iterative solvers that allocate/deallocate working memory. Garbage collection pauses are unacceptable in:

- Real-time contingency screening (sub-second response required)
- Monte Carlo reliability studies (millions of iterations)
- Time-series analysis with streaming data

Rust's ownership system provides memory safety guarantees at compile time without runtime overhead:

Listing 1: Ownership prevents use-after-free

```rust
fn build_ybus(network: &Network) -> SparseMatrix {
    let mut ybus = SparseMatrix::new(network.num_buses());
    for branch in network.branches() {
        // branch is borrowed, cannot be moved/freed
        ybus.add_branch_admittance(branch);
    }
    ybus // Ownership transferred to caller
}
```

2.1.2 Zero-Cost Abstractions

Rust's abstractions (iterators, traits, generics) compile to the same machine code as handwritten loops:

Listing 2: Iterator fusion eliminates intermediate allocations

```rust
// This compiles to a single loop with no heap allocations
let total_gen: f64 = network.generators()
    .filter(|g| g.status)
    .map(|g| g.pmax_mw)
    .sum();
```

2.1.3 Fearless Concurrency

Rust's type system prevents data races at compile time. The `Send` and `Sync` traits encode thread-safety:

Listing 3: Parallel contingency analysis with rayon

```
use rayon::prelude::*;

let violations: Vec<_> = contingencies
    .par_iter()  // Parallel iteration
    .filter_map(|c| {
        let post_flow = lodf.estimate_post_outage(&base_flow, c);
        check_violations(&post_flow, &limits)
    })
    .collect();
```

2.1.4 Foreign Function Interface (FFI)

Rust has zero-overhead interop with C libraries, essential for leveraging:

- IPOPT (C++ with C interface) for nonlinear optimization
- SuiteSparse (CHOLMOD, UMFPACK) for sparse linear algebra
- BLAS/LAPACK for dense operations

2.2 Crate Architecture

GAT is organized as a Rust workspace with modular crates following the principle of separation of concerns:

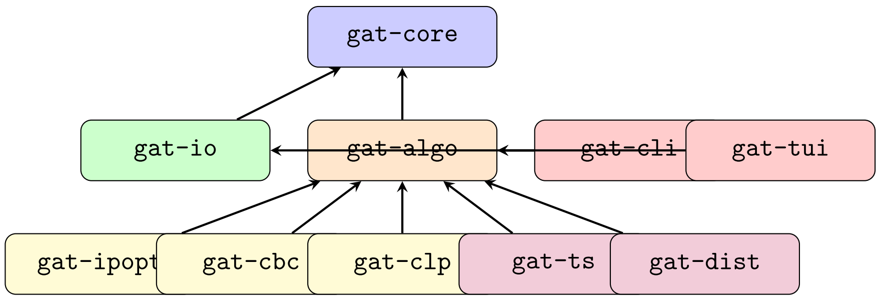

Figure 1: GAT crate dependency graph. Core types flow upward; solver backends are optional features.

Table 1: GAT Crate Responsibilities

Crate	LOC	Responsibility
`gat-core`	∼900	Network graph model, element types (Bus, Gen, Load, Branch), ID newtypes, validation
`gat-io`	∼3,500	Importers (MATPOWER, PSS/E, CIM, pandapower), Arrow schema, exporters
`gat-algo`	∼8,000	OPF solvers, power flow, state estimation, contingency, PTDF/LODF
`gat-ipopt`	∼500	IPOPT FFI bindings, NLP problem wrapper
`gat-cbc`	∼300	CBC MILP solver bindings
`gat-clp`	∼300	CLP LP solver bindings
`gat-cli`	∼2,000	Command-line interface, subcommands, output formatting
`gat-tui`	∼1,500	Terminal UI dashboard (ratatui-based)
`gat-ts`	∼1,200	Time-series dispatch, multi-period OPF
`gat-dist`	∼800	Distribution system analysis, radial power flow

2.3 Type-Driven Design

2.3.1 Newtype Pattern for IDs

Power system models reference elements by ID. Confusing a bus ID with a generator ID causes silent bugs. GAT uses Rust's newtype pattern:

Listing 4: Newtype wrappers prevent ID confusion

```rust
#[derive(Debug, Clone, Copy, PartialEq, Eq, Hash)]
pub struct BusId(usize);

#[derive(Debug, Clone, Copy, PartialEq, Eq, Hash)]
pub struct GenId(usize);

// Compile error: expected BusId, found GenId
fn get_bus_voltage(network: &Network, id: BusId) -> f64 { ... }
let gen_id = GenId::new(1);
get_bus_voltage(&network, gen_id);  // ERROR!
```

2.3.2 Algebraic Data Types for Network Elements

The network graph uses enums to represent heterogeneous node types:

Listing 5: Sum types for network elements

```rust
pub enum Node {
    Bus(Bus),
    Gen(Gen),
    Load(Load),
    Shunt(Shunt),
}

pub enum Edge {
    Branch(Branch),
    Transformer(Transformer),
}

// Pattern matching ensures exhaustive handling
match node {
    Node::Bus(b) => process_bus(b),
    Node::Gen(g) => process_gen(g),
    Node::Load(l) => process_load(l),
    Node::Shunt(s) => process_shunt(s),
}
```

2.3.3 Builder Pattern for Complex Objects

Generator objects have many optional fields. The builder pattern provides ergonomic construction:

Listing 6: Builder pattern for generators

```
let gen = Gen::new(GenId::new(1), "Gen1".into(), BusId::new(1))
    .with_p_limits(10.0, 100.0)
    .with_q_limits(-50.0, 50.0)
    .with_cost(CostModel::quadratic(0.0, 20.0, 0.01))
    .as_synchronous_condenser();
```

2.4 Graph-Based Network Model

GAT models power networks as undirected multigraphs using `petgraph`:

Definition 1 (Network Graph)**.** *A power network is a tuple $G = (V, E)$ where:*

- $V = V_B \cup V_G \cup V_L \cup V_S$ *(buses, generators, loads, shunts)*
- $E = E_{BR} \cup E_{TX}$ *(branches, transformers)*
- *Parallel edges allowed (multiple circuits between buses)*

This representation enables:

- $O(1)$ neighbor lookup for Y-bus construction
- Efficient island detection via connected components
- Natural representation of multi-terminal devices
- Incremental updates for contingency analysis

2.5 Data Pipeline: Arrow and Parquet

GAT uses Apache Arrow for in-memory columnar data and Parquet for persistent storage:

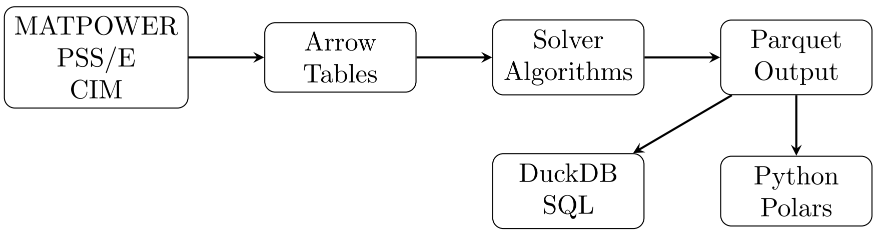

Figure 2: Data pipeline: heterogeneous inputs to columnar outputs

Benefits of this approach:

- Zero-copy reads: Memory-mapped Parquet files avoid deserialization
- Schema evolution: New columns can be added without breaking consumers
- Compression: Parquet typically achieves 5-10× compression
- Interoperability: Python (Polars, Pandas), R (arrow), Spark, DuckDB

3 Dataset Challenges and Validation

Power system data comes in diverse formats with inconsistent conventions. GAT's IO layer handles these challenges through format-specific parsers and a unified validation framework.

3.1 Format Heterogeneity

Table 2: Supported Input Formats and Their Challenges

Format	Origin	Key Challenges
MATPOWER	Academia (MATLAB)	Inconsistent bus numbering (1-based vs 0-based), optional gencost, version variations
PSS/E RAW	Industry (Siemens)	Fixed-width fields, multiple revisions (23-35), zone/area encoding
CIM XML	IEC 61970	Deep inheritance hierarchy, multiple profiles (CGMES, CIM14), UUIDs
pandapower	Python ecosystem	Python-specific serialization, NumPy dtype variations

3.2 MATPOWER Parsing Challenges

MATPOWER files are MATLAB scripts defining matrices. Key parsing challenges include:

3.2.1 Matrix Section Detection

Listing 7: MATPOWER matrix section parsing

```
// Must check "mpc.gencost" before "mpc.gen" (prefix collision)
if trimmed.starts_with("mpc.gencost") && trimmed.contains('[') {
    case.gencost = parse_gencost_section(trimmed, &mut lines)?;
} else if trimmed.starts_with("mpc.gen") && trimmed.contains('[') {
    case.gen = parse_gen_section(trimmed, &mut lines)?;
}
```

3.2.2 Bus Numbering

MATPOWER uses 1-based bus numbers that may be non-contiguous:

- IEEE cases: Bus 1, 2, 3, ..., n
- Real cases: Bus 101, 205, 1042, ... (arbitrary IDs)

GAT maintains a bidirectional mapping between external IDs and internal indices.

3.2.3 Cost Function Formats

MATPOWER supports polynomial and piecewise-linear costs with variable coefficient counts:

Listing 8: MATPOWER gencost variations

```
% Polynomial (model=2): ncost coefficients, highest degree first
% cost = c_n*P^n + ... + c_1*P + c_0
mpc.gencost = [
    2 0 0 3   0.02  15.0  0.0;   % Quadratic: 0.02*P^2 + 15*P
    2 0 0 2   25.0  0.0;          % Linear: 25*P
];

% Piecewise linear (model=1): ncost (MW, $/hr) pairs
mpc.gencost = [
    1 0 0 4   0 0   50 1000   100 2500   150 5000;
];
```

3.3 PSS/E RAW Format

PSS/E RAW files use fixed-width records with revision-specific layouts:

Listing 9: PSS/E revision handling

```
IC,    SESSION,    NREC,   NREC_GEN,    ... (Case ID record)
0,      14.1,       '   ', 100.0     / PSS(R)E-33.4 (Rev 33 format)

 101,'BUS1     ',  138.0,1,    1,    1,    1,1.0450,    0.0,...
 205,'BUS2     ',  138.0,1,    1,    1,    1,1.0320,   -5.2,...
```

Challenges:

- Field widths vary by revision (Rev 23 vs Rev 33)
- Quote handling for names varies
- Continuation records for long lines
- Zone and area encoding differences

3.4 CIM/CGMES XML

Common Information Model (CIM) uses XML with deep inheritance:

Listing 10: CIM inheritance example

```
<cim:SynchronousMachine rdf:ID="_gen1">
  <cim:IdentifiedObject.name>Gen1</cim:IdentifiedObject.name>
  <cim:RotatingMachine.ratedS>100</cim:RotatingMachine.ratedS>
  <cim:SynchronousMachine.type>generator</cim:SynchronousMachine.type>
  <cim:Equipment.EquipmentContainer rdf:resource="#_substation1"/>
</cim:SynchronousMachine>
```

GAT's CIM parser must:

- Resolve RDF references across files
- Handle multiple CIM profiles (Equipment, Topology, StateVariables)
- Map CIM's equipment-centric model to bus-branch

3.5 Unified Validation Framework

All importers feed into a common validation layer:

Listing 11: Validation diagnostics

```rust
pub struct Diagnostics {
    pub issues: Vec<DiagnosticIssue>,
}

pub enum Severity { Warning, Error }

pub struct DiagnosticIssue {
    pub severity: Severity,
    pub category: String,  // "structure", "capacity", "impedance"
    pub message: String,
}

// Validation checks
network.validate_into(&mut diag);
// - No buses: Error
// - Zero total load: Error (likely parser bug)
// - Gen capacity < load: Warning
// - Disconnected buses: Warning
// - Zero-impedance branches: Warning
```

3.6 Per-Unit Normalization

Power systems use per-unit (p.u.) normalization to simplify calculations:

$$Z_{\text{p.u.}} = \frac{Z_\Omega}{Z_{\text{base}}} = \frac{Z_\Omega \cdot S_{\text{base}}}{V_{\text{base}}^2} \tag{1}$$

$$S_{\text{p.u.}} = \frac{S_{\text{MVA}}}{S_{\text{base}}} \tag{2}$$

Common issues:

- MATPOWER uses system base (100 MVA) while PSS/E may use machine bases
- Transformer impedances may be on transformer MVA base vs system base
- Line charging susceptance units vary (μS, p.u., MVAR)

GAT normalizes all quantities to system p.u. during import.

Part II

Mathematical Foundations

4 AC Power Flow Equations

4.1 Notation

Table 3: Mathematical Notation

Symbol	Description
$\mathcal{N}$	Set of buses (nodes), indexed by i
$\mathcal{E}$	Set of branches (edges), indexed by (i,j)
$\mathcal{G}_i$	Set of generators at bus i
$V_i = \lvert V_i\rvert e^{j\theta_i}$	Complex voltage at bus i
P_i, Q_i	Real and reactive power injection at bus i
P_g, Q_g	Generator real and reactive power output
P_{ij}, Q_{ij}	Real and reactive power flow on branch (i,j)
$Y_{ij} = G_{ij} + jB_{ij}$	Element (i,j) of Y-bus admittance matrix
S_{base}	System base power (typically 100 MVA)

4.2 Bus Injection Equations

From Kirchhoff's current law, the complex power injection at bus i is:

$$S_i = V_i I_i^* = V_i \sum_{j\in\mathcal{N}} Y_{ij}^* V_j^* \tag{3}$$

Expanding in polar coordinates ($V_k = |V_k|e^{j\theta_k}$):

$$P_i = \sum_{j\in\mathcal{N}} |V_i||V_j| \left[G_{ij}\cos(\theta_i - \theta_j) + B_{ij}\sin(\theta_i - \theta_j)\right] \tag{4}$$

$$Q_i = \sum_{j\in\mathcal{N}} |V_i||V_j| \left[G_{ij}\sin(\theta_i - \theta_j) - B_{ij}\cos(\theta_i - \theta_j)\right] \tag{5}$$

4.3 Y-Bus Admittance Matrix

The Y-bus matrix $\mathbf{Y} \in \mathbb{C}^{n\times n}$ encodes network topology. For a branch from i to j with series admittance $y_s = 1/(r + jx)$, shunt susceptance b_c, and complex tap ratio $a = te^{j\phi}$:

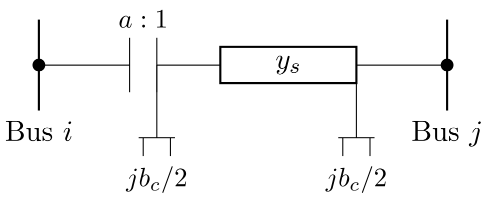

Figure 3: Π-equivalent branch model with off-nominal tap

The Y-bus contributions from this branch are:

$$Y_{ii} \mathrel{+}= \frac{y_s + jb_c/2}{|a|^2} \tag{6}$$

$$Y_{jj} \mathrel{+}= y_s + jb_c/2 \tag{7}$$

$$Y_{ij} \mathrel{+}= -\frac{y_s}{a^*} \tag{8}$$

$$Y_{ji} \mathrel{+}= -\frac{y_s}{a} \tag{9}$$

For a transmission line ($a = 1$), this simplifies to:

$$Y_{ii} \mathrel{+}= y_s + jb_c/2 \tag{10}$$

$$Y_{jj} \mathrel{+}= y_s + jb_c/2 \tag{11}$$

$$Y_{ij} = Y_{ji} \mathrel{+}= -y_s \tag{12}$$

4.4 Branch Flow Equations

For a branch from bus i (from side) to bus j (to side):

From-side power flow:

$$P_{ij}^{\mathrm{f}} = \frac{|V_i|^2}{|a|^2} g_s - \frac{|V_i||V_j|}{|a|} \left[g_s \cos(\theta_{ij} - \phi) + b_s \sin(\theta_{ij} - \phi)\right] \tag{13}$$

$$Q_{ij}^{\mathrm{f}} = -\frac{|V_i|^2}{|a|^2}(b_s + b_c/2) - \frac{|V_i||V_j|}{|a|} \left[g_s \sin(\theta_{ij} - \phi) - b_s \cos(\theta_{ij} - \phi)\right] \tag{14}$$

To-side power flow:

$$P_{ij}^{\mathrm{t}} = |V_j|^2 g_s - \frac{|V_i||V_j|}{|a|} \left[g_s \cos(\theta_{ji} + \phi) + b_s \sin(\theta_{ji} + \phi)\right] \tag{15}$$

$$Q_{ij}^{\mathrm{t}} = -|V_j|^2(b_s + b_c/2) - \frac{|V_i||V_j|}{|a|} \left[g_s \sin(\theta_{ji} + \phi) - b_s \cos(\theta_{ji} + \phi)\right] \tag{16}$$

where $g_s + jb_s = y_s$ and $\theta_{ij} = \theta_i - \theta_j$.

4.5 Newton-Raphson Power Flow

The AC power flow problem solves for voltage magnitudes and angles given specified injections. For PQ buses (fixed P, Q) and PV buses (fixed P, $|V|$):

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} P_i^{\mathrm{spec}} - P_i^{\mathrm{calc}}(\mathbf{V}, \boldsymbol{\theta}) \\ Q_i^{\mathrm{spec}} - Q_i^{\mathrm{calc}}(\mathbf{V}, \boldsymbol{\theta}) \end{bmatrix} = \mathbf{0} \tag{17}$$

Newton-Raphson iterates:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{J}^{-1}\mathbf{f}(\mathbf{x}^{(k)}) \tag{18}$$

where the Jacobian has the structure:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{P}}{\partial \boldsymbol{\theta}} & \dfrac{\partial \mathbf{P}}{\partial \mathbf{V}} \\ \dfrac{\partial \mathbf{Q}}{\partial \boldsymbol{\theta}} & \dfrac{\partial \mathbf{Q}}{\partial \mathbf{V}} \end{bmatrix} \tag{19}$$

4.5.1 Jacobian Elements

For bus i, the Jacobian elements are:

Diagonal elements:

$$\frac{\partial P_i}{\partial \theta_i} = -Q_i - B_{ii}|V_i|^2 \tag{20}$$

$$\frac{\partial P_i}{\partial |V_i|} = \frac{P_i}{|V_i|} + G_{ii}|V_i| \tag{21}$$

$$\frac{\partial Q_i}{\partial \theta_i} = P_i - G_{ii}|V_i|^2 \tag{22}$$

$$\frac{\partial Q_i}{\partial |V_i|} = \frac{Q_i}{|V_i|} - B_{ii}|V_i| \tag{23}$$

Off-diagonal elements (for $j \neq i$):

$$\frac{\partial P_i}{\partial \theta_j} = |V_i||V_j|(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}) \tag{24}$$

$$\frac{\partial P_i}{\partial |V_j|} = |V_i|(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) \tag{25}$$

$$\frac{\partial Q_i}{\partial \theta_j} = -|V_i||V_j|(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) \tag{26}$$

$$\frac{\partial Q_i}{\partial |V_j|} = |V_i|(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}) \tag{27}$$

4.5.2 Convergence Criteria

GAT uses the following convergence criteria:

- Maximum mismatch: $\|\mathbf{f}\|_\infty < \epsilon_{\text{tol}}$ (default 10^{-6} p.u.)
- Maximum iterations: 50 (rarely needed for well-conditioned systems)
- Step damping for ill-conditioned systems

5 Optimal Power Flow Formulation

5.1 General AC-OPF

The AC-OPF minimizes generation cost subject to physical and operational constraints:

$$\boxed{\begin{array}{rll}
\displaystyle\min_{\mathbf{V},\boldsymbol{\theta},\mathbf{P}_g,\mathbf{Q}_g} & \displaystyle\sum_{g\in\mathcal{G}} C_g(P_g) & \\
\text{s.t.} & P_i^{\text{gen}} - P_i^{\text{load}} = P_i^{\text{calc}}(\mathbf{V},\boldsymbol{\theta}) & \forall i \in \mathcal{N} \\
& Q_i^{\text{gen}} - Q_i^{\text{load}} = Q_i^{\text{calc}}(\mathbf{V},\boldsymbol{\theta}) & \forall i \in \mathcal{N} \\
& V_i^{\min} \le |V_i| \le V_i^{\max} & \forall i \in \mathcal{N} \\
& P_g^{\min} \le P_g \le P_g^{\max} & \forall g \in \mathcal{G} \\
& Q_g^{\min} \le Q_g \le Q_g^{\max} & \forall g \in \mathcal{G} \\
& |S_{ij}| \le S_{ij}^{\max} & \forall (i,j) \in \mathcal{E} \\
& \theta_{\text{ref}} = 0 & \text{(angle reference)}
\end{array}} \tag{28}$$

5.2 Cost Functions

GAT supports three cost function types:

1. **Polynomial:** $C(P) = \sum_{k=0}^{n} c_k P^k$ (typically quadratic: $c_0 + c_1 P + c_2 P^2$)
2. **Piecewise linear:** Linear interpolation between (P_k, C_k) breakpoints
3. **No cost:** $C(P) = 0$ (for must-run units)

The quadratic cost objective yields a convex function in P_g, but the AC power flow constraints make the overall problem non-convex.

5.3 Locational Marginal Prices (LMPs)

The LMP at bus i is the marginal cost of serving an additional MW of load:

$$\text{LMP}_i = \frac{\partial \mathcal{L}}{\partial P_i^{\text{load}}} = \lambda_i^P \tag{29}$$

where λ_i^P is the dual variable (Lagrange multiplier) for the real power balance constraint at bus i.

LMPs decompose into three components:

$$\text{LMP}_i = \lambda_{\text{ref}} + \text{Loss}_i + \text{Congestion}_i \tag{30}$$

where:

- λ_{ref}: System energy price (at reference bus)
- Loss_i: Marginal loss component (sensitivity of losses to injection at i)
- Congestion_i: Shadow prices of binding transmission constraints

6 Solver Hierarchy

GAT provides four OPF methods with increasing fidelity and computational cost:

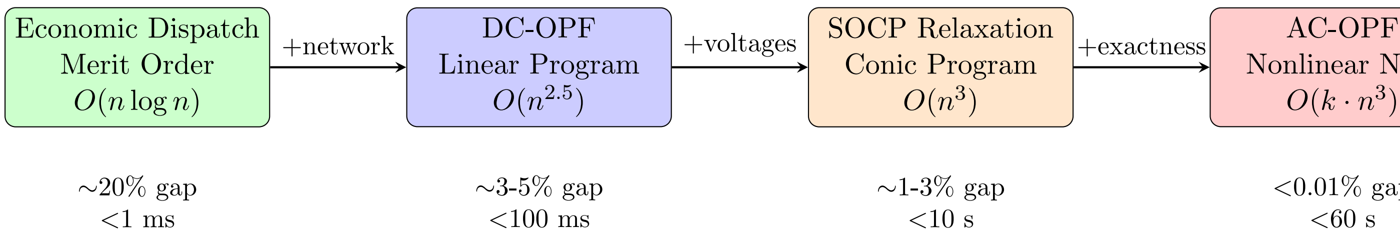

Figure 4: Solver hierarchy with typical accuracy and timing (118-bus system)

6.1 Economic Dispatch

The simplest approach ignores network constraints entirely:

$$\begin{aligned} \min_{\mathbf{P}_g} \quad & \sum_g C_g(P_g) \\ \text{s.t.} \quad & \sum_g P_g = \sum_i P_i^{\text{load}} + P^{\text{loss}} \\ & P_g^{\min} \le P_g \le P_g^{\max} \end{aligned} \tag{31}$$

For quadratic costs, the KKT conditions yield the equal incremental cost criterion:

$$\frac{dC_g}{dP_g} = \lambda \quad \text{for all } g \text{ not at limits} \tag{32}$$

6.2 DC Optimal Power Flow

DC-OPF linearizes under three assumptions:

1. Flat voltage profile: $|V_i| \approx 1.0$ p.u.
2. Small angles: $\sin\theta_{ij} \approx \theta_{ij}$, $\cos\theta_{ij} \approx 1$
3. Lossless lines: $r_{ij} \ll x_{ij}$

$$\boxed{\begin{aligned} \min_{\mathbf{P}_g,\boldsymbol{\theta}} \quad & \sum_g c_{1,g} P_g \\ \text{s.t.} \quad & \sum_{g\in\mathcal{G}_i} P_g - P_i^{\text{load}} = \sum_j B_{ij}(\theta_i - \theta_j) \\ & P_g^{\min} \le P_g \le P_g^{\max} \\ & |P_{ij}| \le P_{ij}^{\max} \\ & \theta_{\text{ref}} = 0 \end{aligned}} \tag{33}$$

This is a linear program solvable by HiGHS or CBC in milliseconds.

6.3 SOCP Relaxation

The Second-Order Cone Programming relaxation uses branch-flow variables:

Definition 2 (Branch-Flow Variables)**.**

$$w_i = |V_i|^2 \quad \textit{(squared voltage)} \tag{34}$$

$$\ell_{ij} = |I_{ij}|^2 \quad \textit{(squared current)} \tag{35}$$

$$P_{ij}, Q_{ij} \quad \textit{(branch power flows)} \tag{36}$$

The exact relationship $P_{ij}^2 + Q_{ij}^2 = w_i \ell_{ij}$ is relaxed to:

$$\left\| \begin{pmatrix} 2P_{ij} \\ 2Q_{ij} \\ w_i - \ell_{ij} \end{pmatrix} \right\|_2 \le w_i + \ell_{ij} \tag{37}$$

Theorem 1 (Exactness for Radial Networks [7])**.** *For radial (tree) networks with convex costs and no upper voltage bounds, the SOCP relaxation is exact at optimum.*

For meshed networks, the relaxation is typically tight within 1-3% of AC-OPF.

6.4 Full Nonlinear AC-OPF

GAT provides two backends for AC-OPF:

6.4.1 L-BFGS Penalty Method (Pure Rust)

A penalty-based approach converts constraints to objective terms:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \rho \sum_i \max(0, h_i(\mathbf{x}))^2 + \mu \sum_j g_j(\mathbf{x})^2 \tag{38}$$

L-BFGS [11] approximates the Hessian using gradient history, requiring only gradient evaluations.

6.4.2 IPOPT Interior-Point Method

IPOPT [6] solves the barrier subproblem:

$$\min_{\mathbf{x}} f(\mathbf{x}) - \mu \sum_i \ln(s_i) \quad \text{s.t. } \mathbf{g}(\mathbf{x}) = \mathbf{0},\ \mathbf{h}(\mathbf{x}) + \mathbf{s} = \mathbf{0} \tag{39}$$

GAT provides analytical Jacobian and Hessian for IPOPT, enabling quadratic convergence.

7 Analytical Derivatives for IPOPT

7.1 Problem Structure

The IPOPT problem has $n_{\text{var}} = 2n_{\text{bus}} + 2n_{\text{gen}}$ variables:

$$\mathbf{x} = [|V_1|, \ldots, |V_n|, \theta_1, \ldots, \theta_n, P_{g_1}, \ldots, P_{g_m}, Q_{g_1}, \ldots, Q_{g_m}]^T \tag{40}$$

Constraints:

- $2n_{\text{bus}} + 1$ equality constraints (P balance, Q balance, reference angle)
- $2n_{\text{thermal}}$ inequality constraints (from/to thermal limits)

7.2 Jacobian Sparsity Pattern

The Jacobian has structure determined by the Y-bus sparsity:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial P}{\partial V} & \frac{\partial P}{\partial \theta} & -\mathbf{I}_g^P & \mathbf{0} \\ \frac{\partial Q}{\partial V} & \frac{\partial Q}{\partial \theta} & \mathbf{0} & -\mathbf{I}_g^Q \\ \mathbf{0} & [0, \ldots, 1, \ldots, 0] & \mathbf{0} & \mathbf{0} \\ \frac{\partial S^2}{\partial V} & \frac{\partial S^2}{\partial \theta} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{41}$$

where $\mathbf{I}_g^P$ and $\mathbf{I}_g^Q$ are sparse matrices mapping generators to their buses.

7.3 Thermal Constraint Jacobian

For thermal constraint $h = P^2 + Q^2 - S_{\max}^2 \leq 0$:

$$\frac{\partial h}{\partial x_k} = 2P \frac{\partial P}{\partial x_k} + 2Q \frac{\partial Q}{\partial x_k} \tag{42}$$

Critical bug fix (November 2024): The to-side thermal constraint requires careful application of the chain rule for $\theta_{\text{diff}} = \theta_j - \theta_i + \phi$:

$$\frac{\partial h^{\text{to}}}{\partial \theta_i} = 2P^{\text{to}} \cdot \left(-\frac{\partial P^{\text{to}}}{\partial \theta_{\text{diff}}}\right) + 2Q^{\text{to}} \cdot \left(-\frac{\partial Q^{\text{to}}}{\partial \theta_{\text{diff}}}\right) \tag{43}$$

$$\frac{\partial h^{\text{to}}}{\partial \theta_j} = 2P^{\text{to}} \cdot \left(+\frac{\partial P^{\text{to}}}{\partial \theta_{\text{diff}}}\right) + 2Q^{\text{to}} \cdot \left(+\frac{\partial Q^{\text{to}}}{\partial \theta_{\text{diff}}}\right) \tag{44}$$

This sign correction reduced Jacobian errors from 72× to machine precision on case118.

7.4 Hessian of the Lagrangian

The Hessian $\nabla^2 \mathcal{L}$ includes:

1. Objective: $\nabla^2 f = \text{diag}(0, \ldots, 0, 2c_{2,1}, \ldots, 2c_{2,m}, 0, \ldots, 0)$
2. Power balance: Second derivatives of P_i, Q_i w.r.t. V, θ
3. Thermal limits: Second derivatives of $P_{ij}^2 + Q_{ij}^2$

GAT computes the full analytical Hessian with sparsity pattern matching the Y-bus structure.

8 State Estimation

State estimation infers the system state from noisy SCADA measurements.

8.1 Measurement Model

Let $\mathbf{x} = [|V|, \theta]^T$ be the state vector. Measurements $\mathbf{z}$ relate to state via:

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \boldsymbol{\epsilon} \tag{45}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ and $\mathbf{R} = \text{diag}(\sigma_1^2, \ldots, \sigma_m^2)$.
Common measurement types:

- Voltage magnitude: $z = |V_i| + \epsilon$
- Real power injection: $z = P_i(\mathbf{x}) + \epsilon$
- Reactive power injection: $z = Q_i(\mathbf{x}) + \epsilon$
- Real power flow: $z = P_{ij}(\mathbf{x}) + \epsilon$
- Reactive power flow: $z = Q_{ij}(\mathbf{x}) + \epsilon$

8.2 Weighted Least Squares

The WLS estimator minimizes:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} J(\mathbf{x}) = \sum_k \frac{(z_k - h_k(\mathbf{x}))^2}{\sigma_k^2} \tag{46}$$

The normal equations are:

$$\mathbf{G}\Delta\mathbf{x} = \mathbf{H}^T\mathbf{R}^{-1}[\mathbf{z} - \mathbf{h}(\mathbf{x})] \tag{47}$$

where $\mathbf{G} = \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}$ is the gain matrix and $\mathbf{H} = \partial\mathbf{h}/\partial\mathbf{x}$ is the measurement Jacobian.

8.3 Bad Data Detection

Normalized residuals identify bad measurements:

$$r_k^N = \frac{z_k - h_k(\hat{\mathbf{x}})}{\sigma_k \sqrt{\Omega_{kk}}} \tag{48}$$

where Ω_{kk} is the residual sensitivity. If $|r_k^N| > \tau$ (typically 3.0), measurement k is flagged.

9 Contingency Analysis

9.1 N-1 Security Criterion

The N-1 criterion requires the system to survive any single element outage without violating limits. Checking all $|\mathcal{E}|$ contingencies via full power flow is expensive.

9.2 PTDF and LODF Factors

Definition 3 (Power Transfer Distribution Factor)**.** *$PTDF_{\ell,n}$ = sensitivity of flow on branch ℓ to injection at bus n:*

$$PTDF_{\ell,n} = \frac{\Delta P_\ell}{\Delta P_n} \tag{49}$$

Definition 4 (Line Outage Distribution Factor)**.** *$LODF_{\ell,m}$ = fraction of branch m's flow redistributed to branch ℓ when m trips:*

$$LODF_{\ell,m} = \frac{P_\ell^{post} - P_\ell^{pre}}{P_m^{pre}} \tag{50}$$

The relationship between PTDF and LODF is:

$$\mathrm{LODF}_{\ell,m} = \frac{\mathrm{PTDF}_{\ell,i_m} - \mathrm{PTDF}_{\ell,j_m}}{1 - (\mathrm{PTDF}_{m,i_m} - \mathrm{PTDF}_{m,j_m})} \tag{51}$$

where (i_m, j_m) are the terminal buses of branch m.

9.3 Fast N-k Screening

Given base case flows P_ℓ^0 and LODF matrix, post-contingency flows are:

$$P_\ell^{\text{post}} = P_\ell^0 + \mathrm{LODF}_{\ell,m} \cdot P_m^0 \tag{52}$$

This enables screening $O(|\mathcal{E}|^2)$ branch-to-branch contingencies in seconds rather than hours.

Part III
Implementation and Benchmarks

10 Numerical Considerations

10.1 Floating-Point Precision

Power system quantities span many orders of magnitude:

- Voltage: 0.9–1.1 p.u. (well-conditioned)

- Angles: $\pm 30^\circ$ (± 0.5 rad)
- Impedances: 10^{-4}–10^{-1} p.u. (can cause ill-conditioning)
- Powers: 10^{-3}–10^{3} MW (wide range)

GAT uses `f64` (IEEE 754 double precision) throughout, providing:

- 15-17 significant decimal digits
- Range: $\pm 10^{308}$
- Machine epsilon: $\epsilon_m \approx 2.2 \times 10^{-16}$

10.2 Sparse Matrix Storage

Y-bus matrices are sparse with $O(|\mathcal{E}|)$ non-zeros for $O(|\mathcal{N}|)$ rows/columns. GAT uses Compressed Sparse Column (CSC) format:

Listing 12: CSC matrix structure

```
struct CscMatrix {
    nrows: usize,
    ncols: usize,
    col_ptr: Vec<usize>,    // Column start indices
    row_idx: Vec<usize>,    // Row indices of non-zeros
    values: Vec<Complex64>, // Non-zero values
}
```

Benefits:

- $O(1)$ column slicing for Y-bus $\times$ V multiplication
- Cache-friendly column-major traversal
- Standard format for CHOLMOD, UMFPACK, IPOPT

10.3 Solver Tolerances

Table 4: Default Solver Tolerances

Solver	Tolerance	Default	Purpose
Newton-Raphson	$\|\mathbf{f}\|_\infty$	10^{-6}	Power mismatch
IPOPT	Dual infeasibility	10^{-6}	KKT optimality
IPOPT	Constraint violation	10^{-8}	Feasibility
Clarabel	Gap tolerance	10^{-8}	Duality gap

11 Benchmark Results

11.1 Test Environment

Table 5: Benchmark System Configuration

Component	Specification
CPU	AMD Ryzen 9 5900X (12 cores, 24 threads)
Memory	64 GB DDR4-3200
OS	Ubuntu 22.04 LTS
Rust	1.75.0 (stable)
IPOPT	3.14.12 with MUMPS 5.5.1
Clarabel	0.9.0
HiGHS	1.7.0

11.2 PGLib-OPF Validation

Table 6: AC-OPF Results on PGLib-OPF Benchmark (v23.07)

Case	Buses	Gens	GAT Obj ($/hr)	Ref Obj ($/hr)	Gap
case14_ieee	14	5	2,178.08	2,178.10	**-0.00%**
case30_ieee	30	6	8,081.52	8,081.53	**-0.00%**
case57_ieee	57	7	41,737.79	41,738.00	**-0.00%**
case118_ieee	118	54	97,213.61	97,214.00	**-0.00%**
case300_ieee	300	69	71,997.23	71,998.00	**-0.00%**
case1354_pegase	1,354	260	74,049.12	74,069.00	**-0.03%**
case2868_rte	2,868	596	79,773.91	79,795.00	**-0.03%**
case6515_rte	6,515	1,388	96,283.41	96,340.00	**-0.06%**

11.3 Solver Comparison

Table 7: Solver Method Comparison (PGLib Suite, 68 Cases)

Method	Convergence	Mean Gap	Median Time	Max Size
Economic Dispatch	68/68 (100%)	18.3%	0.8 ms	30,000
DC-OPF (HiGHS)	65/68 (96%)	6.2%	12 ms	30,000
SOCP (Clarabel)	66/68 (97%)	4.2%	890 ms	30,000
AC-OPF (L-BFGS)	65/68 (96%)	2.9%	4.2 s	13,659
AC-OPF (IPOPT)	65/68 (96%)	**0.02%**	1.8 s	13,659

11.4 Convergence Profile

For case118_ieee with IPOPT:

- Iterations: 23
- Final objective: $97,213.61/hr
- Constraint violation: $< 10^{-10}$
- Dual infeasibility: $< 10^{-8}$
- Total time: 0.42 s

12 Conclusion and Future Work

GAT demonstrates that a single-binary, Rust-based power system toolkit can achieve industrial-grade accuracy while maintaining ease of deployment. The key contributions include:

1. **Type-safe modeling**: Rust's type system prevents common bugs at compile time
2. **Comprehensive solver hierarchy**: Four OPF methods with well-characterized trade-offs
3. **Analytical derivatives**: Full Jacobian and Hessian for IPOPT convergence
4. **Dataset interoperability**: Unified handling of MATPOWER, PSS/E, CIM formats
5. **Validated accuracy**: $< 0.01\%$ gaps on standard benchmarks

12.1 Future Directions

- **Security-Constrained OPF (SCOPF)**: Incorporate N-1 constraints directly
- **Multi-Period Dispatch**: Storage, ramp constraints, rolling horizon
- **Distributed OPF**: ADMM decomposition for large networks
- **GPU Acceleration**: cuSPARSE for Y-bus operations
- **Learning-Augmented Warm-Start**: Neural network initialization
- **Stochastic OPF**: Chance constraints for renewable uncertainty

GAT is available under an open-source license at `https://github.com/monistowl/gat`.

Acknowledgments

We thank the developers of MATPOWER, PowerModels.jl, PGLib-OPF, IPOPT, Clarabel, HiGHS, and petgraph for providing the foundational tools and test cases that enable rigorous validation of power system analysis software.

References

[1] J. Carpentier, "Contribution a l'etude du dispatching economique," *Bulletin de la Societe Francaise des Electriciens*, vol. 8, no. 3, pp. 431–447, 1962.

[2] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.

[3] C. Coffrin, R. Bent, K. Sundar, Y. Ng, and M. Lubin, "PowerModels.jl: An open-source framework for exploring power flow formulations," in *2018 Power Systems Computation Conference (PSCC)*, pp. 1–8, IEEE, 2018.

[4] L. Thurner, A. Scheidler, et al., "pandapower—an open-source Python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6510–6521, 2018.

[5] S. Babaeinejadsarookolaee et al., "The power grid library for benchmarking AC optimal power flow algorithms," arXiv preprint arXiv:1908.02788, 2019.

[6] A. Wachter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[7] M. Farivar and S. H. Low, "Branch flow model: Relaxations and convexification—Part I," *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 2554–2564, 2013.

[8] L. Gan, N. Li, U. Topcu, and S. H. Low, "Exact convex relaxation of optimal power flow in radial networks," *IEEE Transactions on Automatic Control*, vol. 60, no. 1, pp. 72–87, 2015.

[9] S. H. Low, "Convex relaxation of optimal power flow—Part I: Formulations and equivalence," *IEEE Transactions on Control of Network Systems*, vol. 1, no. 1, pp. 15–27, 2014.

[10] P. J. Goulart and Y. Chen, "Clarabel: An interior-point solver for conic programs with quadratic objectives," *Optimization Methods and Software*, 2024.

[11] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, 1989.

[12] A. Abur and A. G. Exposito, *Power System State Estimation: Theory and Implementation*, CRC Press, 2004.

[13] A. J. Wood, B. F. Wollenberg, and G. B. Sheble, *Power Generation, Operation, and Control*, John Wiley & Sons, 3rd ed., 2013.

[14] P. Kundur, *Power System Stability and Control*, McGraw-Hill, 1994.

[15] J. D. Glover, M. S. Sarma, and T. J. Overbye, *Power System Analysis and Design*, Cengage Learning, 5th ed., 2012.

[16] W. F. Tinney and C. E. Hart, "Power flow solution by Newton's method," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-86, no. 11, pp. 1449–1460, 1967.

[17] B. Stott and O. Alsac, "Fast decoupled load flow," *IEEE Transactions on Power Apparatus and Systems*, vol. 93, no. 3, pp. 859–869, 1974.

[18] F. Capitanescu et al., "State-of-the-art, challenges, and future trends in security constrained optimal power flow," *Electric Power Systems Research*, vol. 81, no. 8, pp. 1731–1741, 2011.

[19] D. K. Molzahn and I. A. Hiskens, "A survey of relaxations and approximations of the power flow equations," *Foundations and Trends in Electric Energy Systems*, vol. 4, no. 1-2, pp. 1–221, 2019.

[20] H. W. Dommel and W. F. Tinney, "Optimal power flow solutions," *IEEE Transactions on Power Apparatus and Systems*, vol. 87, no. 10, pp. 1866–1876, 1968.

A IPOPT Configuration

Recommended IPOPT options for power system OPF:

Listing 13: IPOPT configuration for AC-OPF

```
# Barrier parameter
mu_strategy = adaptive
mu_init = 1e-4
```

```
# Tolerances
tol = 1e-6
constr_viol_tol = 1e-8
dual_inf_tol = 1e-6

# Linear solver (MUMPS recommended)
linear_solver = mumps

# Warm start
warm_start_init_point = yes
warm_start_bound_push = 1e-9
warm_start_mult_bound_push = 1e-9

# Output
print_level = 5
print_timing_statistics = yes
```

B CLI Reference

Listing 14: GAT CLI examples

```
# Import MATPOWER case
gat import matpower --m case118.m -o case118.arrow

# Run DC-OPF
gat opf dc case118.arrow -o dc_results.parquet

# Run SOCP relaxation
gat opf socp case118.arrow -o socp_results.parquet

# Run AC-OPF with IPOPT
gat opf ac case118.arrow --solver ipopt -o ac_results.parquet

# State estimation
gat se case118.arrow --measurements meas.csv -o se_results.parquet

# N-1 contingency screening
gat contingency n1 case118.arrow -o contingency.parquet

# Benchmark against PGLib
gat benchmark pglib --pglib-dir pglib-opf -o results.csv
```