1. Governing idea (one sentence)

> **Adjust nodal pressures so that the mass imbalance at every node goes to zero**, using how each connected branch flow responds to pressure.

That's it. Everything below just makes that precise.

---

2. Network definitions

- Nodes: $i = 1 \dots N$
- Branch k connects nodes $i \to j$
- Flow sign convention:
  $Q_k > 0$ means flow from i to j

Branch flow law (generic, nonlinear)

$$Q_k = f_k(\Delta p_k), \quad \Delta p_k = p_i - p_j - \Delta p_k^{\text{jump}}$$

Examples:

- Linear: $Q = K \Delta p$
- Quadratic: $Q = C \, \text{sign}(\Delta p) \quad |\Delta p|$

---

3. Mass imbalance at a node

For node i:

$$R_i = \sum_{k \in i} s_{ik} Q_k + S_i$$

Where:

- $s_{ik} = +1$ if branch enters node
- $s_{ik} = -1$ if branch leaves node
- S_i = source/sink (fixed inflow/outflow)

Goal:

$$R_i = 0 \quad \forall i$$

---

4. Why pressure correction is needed

You already have a guessed pressure field p_i.

Flows computed from it **do not satisfy mass balance**:

$$R_i \neq 0$$

So we apply a correction:

$$p_i^{new} = p_i + p_i'$$

This induces flow corrections Q_k'.

---

5. Linearizing branch flow (key step)

For branch $k : i \to j$:

$$Q_k' = \frac{dQ_k}{d(\Delta p_k)} (p_i' - p_j')$$

Define **branch conductance**

$$G_k = \left. \frac{dQ_k}{d(\Delta p_k)} \right|_{\text{current}}$$

Examples:

- $Q = K\Delta p \Rightarrow G = K$
- $Q = C \quad |\Delta p| \Rightarrow G = \frac{C}{2 \quad |\Delta p|}$

---

6. Corrected mass balance at node

Corrected imbalance:

$$R_i + R_i' = 0$$

Where:

$$R_i' = \sum_{k \in i} s_{ik} Q_k'$$

Substitute Q_k':

$$\sum_{k \in i} G_k (p_i' - p_j') = -R_i$$

7. Node-wise pressure correction equation

Rearrange:

$$\boxed{p_i' \sum_{k \in i} G_k \; - \sum_{j \in \mathrm{nbr}(i)} G_{ij} p_j' = -R_i}$$

This equation is obtained **purely from mass imbalance**.

---

8. Explicit pressure correction (no matrix solve)

If you **lag neighbor corrections** (Gauss–Seidel style):

$$\boxed{p_i' = \frac{-R_i + \sum_j G_{ij} p_j'}{\sum_{k \in i} G_k}}$$

This is the **explicit pressure update formula** you were asking for.

---

9. Algorithm summary (mass-imbalance solver)

1. Guess nodal pressures p_i
2. Compute branch flows Q_k
3. Compute node mass imbalance R_i
4. Compute conductances G_k
5. Update pressures using:

$$p_i \leftarrow p_i + \alpha p_i'$$

6. Repeat until $|R_i| < \varepsilon$

No SIMPLE.

No global Newton matrix.

Only **mass balance + local flow sensitivity**.

---

10. Why this works (intuition)

- Mass imbalance tells you **how wrong the pressure is**

- Conductance tells you **how much flow reacts to pressure**
- The correction redistributes pressure until all imbalances vanish

This is essentially **Kirchhoff's law with nonlinear resistors**, solved iteratively.

1. Model assumptions (explicit & simple)

Branch types

- **PIPE**

$$\Delta p = RQ|Q| \quad \Rightarrow \quad Q = \text{sign}(\Delta p) \quad \frac{|\Delta p|}{R}$$

- **FAN / pressure jump**

$$\Delta p_{\text{fan}} = a + bQ + cQ^2$$

  Used **inside the branch**, not as a node equation.

---

2. Core data structures

```
enum BranchType {
    PIPE,
    FAN
}

class Node {
    int id;
    double pressure;
    double massImbalance; // R_i
    boolean isReference = false;
}

class Branch {
    int from;
    int to;
    BranchType type;

    // pipe resistance
    double R;

    // fan coefficients: dp = a + bQ + cQ^2
    double a, b, c;

    double flow;
    double conductance;
}
```

---

3. Branch flow evaluation (with pressure jump)

```
static void computeBranchFlow(Branch br, Node ni, Node nj) {

    double dp = ni.pressure - nj.pressure;

    if (br.type == BranchType.PIPE) {
        double sign = Math.signum(dp);
        br.flow = sign * Math.sqrt(Math.abs(dp) / br.R);
    }

    if (br.type == BranchType.FAN) {
```

```
            // Solve: dp = a + bQ + cQ^2
            double A = br.c;
            double B = br.b;
            double C = br.a - dp;

            double disc = B * B - 4 * A * C;
            disc = Math.max(disc, 0.0);

            // choose physically consistent root
            br.flow = (-B + Math.sqrt(disc)) / (2 * A);
        }
    }
```

---

4. Branch conductance (dQ / dΔp)

```
    static void computeConductance(Branch br, Node ni, Node nj) {

        double dp = ni.pressure - nj.pressure;
        double eps = 1e-8;

        if (br.type == BranchType.PIPE) {
            br.conductance = 1.0 / (2.0 * Math.sqrt(br.R * Math.max(Math.abs(dp
        }

        if (br.type == BranchType.FAN) {
            // From: dp = a + bQ + cQ^2
            // dQ/ddp = 1 / (b + 2cQ)
            br.conductance = 1.0 / (br.b + 2.0 * br.c * br.flow);
        }
    }
```

---

5. Mass imbalance at nodes (only physics used)

```
    static void computeMassImbalance(Node[] nodes, Branch[] branches) {

        for (Node n : nodes) {
            n.massImbalance = 0.0;
        }
```

```
        for (Branch br : branches) {
            nodes[br.from].massImbalance -= br.flow;
            nodes[br.to].massImbalance   += br.flow;
        }
    }
```

---

6. Explicit pressure correction update

This is the **key formula derived purely from mass balance**:

$$p'_i = \frac{-R_i + \sum G_{ij} p'_j}{\sum G_{ik}}$$

Implemented explicitly:

```
    static void pressureCorrection(
            Node[] nodes,
            Branch[] branches,
            double relaxation) {

        int n = nodes.length;
        double[] dp = new double[n];

        for (int i = 0; i < n; i++) {

            Node ni = nodes[i];
            if (ni.isReference) continue;

            double sumG = 0.0;
            double rhs  = -ni.massImbalance;

            for (Branch br : branches) {

                if (br.from == i || br.to == i) {

                    int j = (br.from == i) ? br.to : br.from;
                    Node nj = nodes[j];

                    double G = br.conductance;
                    sumG += G;
```

```
                rhs += G * dp[j];
            }
        }

        dp[i] = rhs / sumG;
    }

    for (int i = 0; i < n; i++) {
        if (!nodes[i].isReference) {
            nodes[i].pressure += relaxation * dp[i];
        }
    }
}
```

---

7. Solver loop (complete algorithm)

```
static void solve(
        Node[] nodes,
        Branch[] branches,
        int maxIter,
        double tol) {

    double relax = 0.7;

    for (int iter = 0; iter < maxIter; iter++) {

        // 1. Compute flows
        for (Branch br : branches) {
            computeBranchFlow(br, nodes[br.from], nodes[br.to]);
        }

        // 2. Conductances
        for (Branch br : branches) {
            computeConductance(br, nodes[br.from], nodes[br.to]);
        }

        // 3. Mass imbalance
        computeMassImbalance(nodes, branches);

        // 4. Check convergence
```

```
            double maxR = 0.0;
            for (Node n : nodes) {
                maxR = Math.max(maxR, Math.abs(n.massImbalance));
            }

            if (maxR < tol) {
                System.out.println("Converged in " + iter + " iterations");
                return;
            }

            // 5. Pressure correction
            pressureCorrection(nodes, branches, relax);
        }

        System.out.println("Did not converge");
    }
```

---

8. Why this is not SIMPLE

- No guessed velocities
- No pressure correction equation from momentum
- No matrix solve
- Only:
  - nonlinear branch laws
  - nodal mass imbalance
  - local linear sensitivity

This is essentially **Kirchhoff + nonlinear resistors + pressure sources**.

1. CSV formats (simple & explicit)

nodes.csv

```
id,pressure,isReference
0,0.0,true
1,0.0,false
2,0.0,false
```

- One node **must** be reference
- Initial pressure is just a guess

---

branches.csv

```
from,to,type,R,a,b,c,Qfixed
0,1,PIPE,100.0,0,0,0,0
1,2,FAN,0,50.0,-2.0,0.1,0
2,0,FIXED_FLOW,0,0,0,0,0.25
```

Interpretation

type	meaning
PIPE	$\Delta p = R\,Q$
FAN	$\Delta p = a + bQ + cQ^2$
FIXED_FLOW	$Q = Q_{fixed}$ (independent of pressure)

---

2. Branch type extension

```
enum BranchType {
    PIPE,
    FAN,
    FIXED_FLOW
}
```

---

3. CSV reader utilities

```
import java.io.*;
import java.util.*;

static Node[] readNodes(String file) throws Exception {

    List<Node> list = new ArrayList<>();
    BufferedReader br = new BufferedReader(new FileReader(file));
    br.readLine(); // header

    String line;
    while ((line = br.readLine()) != null) {
        String[] t = line.split(",");
        Node n = new Node();
        n.id = Integer.parseInt(t[0]);
        n.pressure = Double.parseDouble(t[1]);
        n.isReference = Boolean.parseBoolean(t[2]);
        list.add(n);
    }
    br.close();
    return list.toArray(new Node[0]);
}


static Branch[] readBranches(String file) throws Exception {

    List<Branch> list = new ArrayList<>();
    BufferedReader br = new BufferedReader(new FileReader(file));
    br.readLine(); // header

    String line;
    while ((line = br.readLine()) != null) {
        String[] t = line.split(",");

        Branch b = new Branch();
        b.from = Integer.parseInt(t[0]);
        b.to   = Integer.parseInt(t[1]);
        b.type = BranchType.valueOf(t[2]);

        b.R = Double.parseDouble(t[3]);
        b.a = Double.parseDouble(t[4]);
```

```
            b.b = Double.parseDouble(t[5]);
            b.c = Double.parseDouble(t[6]);
            b.flow = Double.parseDouble(t[7]); // Qfixed for FIXED_FLOW

            list.add(b);
        }
        br.close();
        return list.toArray(new Branch[0]);
    }
```

---

4. Branch flow computation (updated)

```
static void computeBranchFlow(Branch br, Node ni, Node nj) {

    double dp = ni.pressure - nj.pressure;

    switch (br.type) {

        case PIPE:
            double sign = Math.signum(dp);
            br.flow = sign * Math.sqrt(Math.abs(dp) / br.R);
            break;

        case FAN:
            // dp = a + bQ + cQ^2
            double A = br.c;
            double B = br.b;
            double C = br.a - dp;

            double disc = Math.max(B * B - 4 * A * C, 0.0);
            br.flow = (-B + Math.sqrt(disc)) / (2 * A);
            break;

        case FIXED_FLOW:
            // flow is prescribed
            // positive from -> to
            // already stored in br.flow
            break;
    }
}
```

---

5. Conductance calculation (important detail)

Fixed-flow branches **must not influence pressure correction**.

```
static void computeConductance(Branch br, Node ni, Node nj) {

    double dp = ni.pressure - nj.pressure;
    double eps = 1e-8;

    switch (br.type) {

        case PIPE:
            br.conductance =
                1.0 / (2.0 * Math.sqrt(br.R * Math.max(Math.abs(dp), eps))
            break;

        case FAN:
            br.conductance =
                1.0 / (br.b + 2.0 * br.c * br.flow);
            break;

        case FIXED_FLOW:
            br.conductance = 0.0; // 🔑 critical
            break;
    }
}
```

---

6. Mass imbalance (unchanged physics)

Fixed-flow branches contribute **directly** to imbalance — exactly what we want.

```
static void computeMassImbalance(Node[] nodes, Branch[] branches) {

    for (Node n : nodes) {
        n.massImbalance = 0.0;
    }

    for (Branch br : branches) {
        nodes[br.from].massImbalance -= br.flow;
        nodes[br.to].massImbalance   += br.flow;
    }
}
```

---

7. Pressure correction (unchanged)

Because `G = 0` for fixed-flow branches, they:

- affect $\mathbf{R_i}$
- do **not** affect pressure coupling

This is mathematically consistent.

```
  static void pressureCorrection(
          Node[] nodes,
          Branch[] branches,
          double relaxation) {

      int n = nodes.length;
      double[] dp = new double[n];

      for (int i = 0; i < n; i++) {

          Node ni = nodes[i];
          if (ni.isReference) continue;

          double sumG = 0.0;
          double rhs  = -ni.massImbalance;

          for (Branch br : branches) {
              if (br.conductance == 0.0) continue;

              if (br.from == i || br.to == i) {
                  int j = (br.from == i) ? br.to : br.from;
                  sumG += br.conductance;
                  rhs  += br.conductance * dp[j];
              }
          }

          dp[i] = rhs / sumG;
      }

      for (int i = 0; i < n; i++) {
          if (!nodes[i].isReference) {
              nodes[i].pressure += relaxation * dp[i];
          }
      }
```

```
}
```

---

8. Main driver

```
public static void main(String[] args) throws Exception {

    Node[] nodes = readNodes("nodes.csv");
    Branch[] branches = readBranches("branches.csv");

    solve(nodes, branches, 5000, 1e-6);

    for (Node n : nodes) {
        System.out.printf("Node %d : p = %.6f%n", n.id, n.pressure);
    }
}
```

---

9. Why this is correct physics

Feature	Treatment
Pipes	nonlinear resistance
Fans	internal pressure source
Fixed flows	pure mass source
Pressure	adjusted only to fix mass
Reference	removes singularity

This is exactly how **industrial HVAC** / **hydraulic solvers** work internally — just without the matrix machinery.