# Informe Técnico Exhaustivo: Refactorización Homológica y Algebraica del Módulo 'flux_condenser.py' hacia una Bomba Hidráulica Lineal

## Resumen Ejecutivo

Este documento constituye una investigación técnica integral y un plan de ingeniería detallado para la refactorización crítica del código heredado flux_condenser.py. El objetivo es transformar la lógica fenomenológica existente —presumiblemente basada en la acumulación de flujo o "condensación" de variables de estado— en un modelo de simulación física rigurosa de una **Bomba Hidráulica Lineal**. Esta transformación se fundamenta en la convergencia interdisciplinaria de tres dominios teóricos avanzados: la **Física de Circuitos** (para establecer las leyes constitutivas mediante la analogía electro-hidráulica), la **Topología Algebraica** (para definir la conectividad de la red y las leyes de conservación a través de la homología simplicial), y el **Álgebra Lineal** (para la resolución numérica del sistema resultante de ecuaciones diferenciales parciales discretizadas).
El análisis establece que una bomba hidráulica lineal opera isomórficamente a una fuente de corriente ideal en un circuito eléctrico. Al mapear la presión hidráulica al potencial eléctrico y el caudal volumétrico a la corriente eléctrica, el proceso de refactorización traslada el sistema de un modelo de "flujo" no restringido a una red de flujo conservativa gobernada por la ecuación de Poisson discreta sobre grafos. Las recomendaciones arquitectónicas aprovechan la pila científica de Python contemporánea (2025), específicamente **NetworkX** para la gestión de grafos topológicos y **SciPy/NumPy** para operaciones de matrices dispersas, encapsuladas dentro de un Diseño Orientado a Objetos (OOD) robusto que prioriza la seguridad de tipos y la modularidad científica.

---

## 1. Fundamentos Teóricos: La Analogía Electro-Hidráulica Rigurosa

La refactorización del módulo flux_condenser.py exige un abandono de la lógica ad-hoc en favor de un formalismo físico estricto. Para modelar una bomba hidráulica lineal con la precisión requerida por sistemas de ingeniería modernos, es imperativo establecer una correspondencia biyectiva entre el dominio de la mecánica de fluidos incompresibles y la

teoría de circuitos lineales concentrados. Esta correspondencia no es meramente heurística; es un isomorfismo matemático que permite la aplicación directa de los teoremas de conservación de energía y carga al transporte de fluidos.

## 1.1 Isomorfismo de Variables de Estado: Potencial y Flujo

La base de la refactorización reside en la identificación correcta de las variables de estado del sistema. En la física de sistemas dinámicos, las variables se categorizan fundamentalmente en variables "a través" (through variables) y variables "a través de" o "transversales" (across variables). Esta distinción es crucial para la correcta aplicación del álgebra lineal y la topología en etapas posteriores del desarrollo.[1]

### 1.1.1 La Variable Transversal: Presión como Potencial Generalizado

En el modelo hidráulico objetivo, la **Presión ($P$)** se define como la variable transversal, análoga al **Voltaje ($V$)** o potencial eléctrico en la teoría de circuitos.[2]

- **Análisis Dimensional y Energético:** La presión se mide en Pascales ($Pa$), que dimensionalmente equivale a Newtons por metro cuadrado ($N/m^2$) o, más relevantemente para nuestra analogía energética, Julios por metro cúbico ($J/m^3$). Esto representa la densidad de energía potencial almacenada en el fluido por unidad de volumen. Este concepto es directamente paralelo al Voltaje ($V$), que se define como Julios por Coulomb ($J/C$), representando la energía potencial por unidad de carga eléctrica.[1]
- **Rol Topológico (Cadenas y Cocadenas):** Desde la perspectiva de la topología algebraica, las variables de potencial como la presión se asocian intrínsecamente a los vértices (0-simplices) del grafo de la red. Matemáticamente, la presión se modela como una **0-cocadena** ($c^0$), una función que asigna un valor escalar real a cada nodo de la red.[3]
- **Implicación para la Refactorización:** El código original flux_condenser.py probablemente trataba el "flujo" o la "energía" como una cantidad absoluta. El nuevo modelo debe tratar la presión como una cantidad relativa. No existe una presión absoluta en la formulación dinámica sin una referencia; por lo tanto, el sistema requiere la definición explícita de un nodo de tierra (ground node) o reservorio de referencia donde $P = 0$.[4]

### 1.1.2 La Variable de Flujo: Caudal Volumétrico como Corriente

El **Caudal Volumétrico ($Q$)** se establece como la variable "a través", análoga a la **Corriente Eléctrica ($I$)**.[1]

- **Análisis Dimensional:** El caudal se mide en metros cúbicos por segundo ($m^3/s$). Su contraparte eléctrica, la corriente, se mide en Coulombs por segundo ($C/s$ o Amperios). Ambas representan la tasa de transporte de una cantidad conservada (volumen de fluido incompresible o carga eléctrica) a través de una sección transversal.[2]
- **Rol Topológico:** A diferencia de la presión, el flujo es una cantidad direccional asociada

a los enlaces o aristas (1-simplices) de la red. Se modela como una **1-cadena** ($c_1$). La orientación es fundamental: un flujo positivo en una dirección $u \to v$ es matemáticamente equivalente a un flujo negativo en la dirección $v \to u$. El código refactorizado debe implementar estructuras de datos que respeten rigurosamente esta orientación vectorial.[5]

## 1.2 Leyes Constitutivas y el Régimen de Linealidad

El término "Condensador de Flujo" en el código original sugiere un comportamiento capacitivo (almacenamiento de energía en campo). Sin embargo, el objetivo de la refactorización es una "Bomba Hidráulica Lineal", lo que implica un elemento activo generador de flujo. Para que la simulación sea "lineal" y susceptible de ser resuelta mediante álgebra lineal matricial directa (sin iteraciones de Newton-Raphson), debemos asumir un régimen de flujo específico.

### 1.2.1 Resistencia Hidráulica y la Ley de Ohm

En la teoría de circuitos, la Ley de Ohm ($V = IR$) relaciona el potencial y el flujo a través de una constante de proporcionalidad (Resistencia). En hidráulica, la relación general entre la caída de presión ($\Delta P$) y el caudal ($Q$) suele ser cuadrática debido a la turbulencia (Ecuación de Darcy-Weisbach: $\Delta P \propto Q^2$). No obstante, para cumplir con el requisito de "bomba lineal" y "álgebra lineal", el sistema debe modelarse en el régimen de **Flujo de Hagen-Poiseuille** (flujo laminar en tuberías).[2]

- **Ecuación de Poiseuille:** $\Delta P = R_{h} \cdot Q$.
- **Definición de Resistencia ($R_h$):** Para una tubería cilíndrica de longitud $L$ y radio $r$, la resistencia hidráulica lineal es $R_h = \frac{8 \mu L}{\pi r^4}$, donde $\mu$ es la viscosidad dinámica del fluido.
- **Estrategia de Código:** El refactoring debe reemplazar cualquier lógica de "condensación" no lineal con una matriz de conductancia constante. Si el código original flux_condenser.py calculaba la acumulación de flujo basada en umbrales complejos, estos deben ser reemplazados por la relación lineal instantánea definida por la matriz de resistencia.[6]

## 1.3 Comparativa Estructural de Variables

La siguiente tabla resume la transformación ontológica de las variables del código legado al nuevo paradigma físico:

| Concepto Físico | Dominio Eléctrico (Circuito) | Dominio Hidráulico (Objetivo) | Representación Matemática | Rol en Refactorización |
|---|---|---|---|---|
| **Potencial** | Voltaje ($V$, Volts) | Presión ($P$, Pascales) | 0-cocadena ($C^0$) | Incógnita a resolver ($x$ en $Ax=b$) |
| **Flujo** | Corriente ($I$, Amperios) | Caudal ($Q$, $m^3/s$) | 1-cadena ($C_1$) | Variable derivada o fuente ($b$) |

| | | | | |
|---|---|---|---|---|
| **Carga** | Carga ($q$, Coulombs) | Volumen ($Vol$, $m^3$) | 0-cadena ($C_0$) | Integral del flujo en el tiempo |
| **Resistencia** | Resistencia ($R$, Ohms) | Resistencia Hidráulica ($R_h$) | Operador escalar en aristas | Coeficiente en Matriz Laplaciana |
| **Fuente Activa** | Fuente de Corriente Ideal | Bomba de Desplazamiento Positivo | Divergencia de Flujo Prescrita | Vector de Términos Independientes ($s$) |
| **Almacenamiento** | Capacitancia ($C$, Faradios) | Acumulador / Tanque ($C_h$) | Coeficiente diferencial ($d/dt$) | Eliminado en análisis estático |

---

# 2. Topología Algebraica Aplicada a Redes de Flujo

Para satisfacer el requisito de aplicar "Topología Algebraica", la refactorización no puede limitarse a usar arrays bidimensionales simples. Debe formalizar la red hidráulica como un **Complejo Simplicial**. Este enfoque permite derivar las leyes de conservación (Kirchhoff) directamente de las propiedades topológicas del espacio, garantizando la robustez matemática del simulador.[3]

## 2.1 El Complejo Simplicial de la Red Hidráulica

La red se modela como un grafo dirigido $G = (V, E)$, que constituye un complejo simplicial de dimensión 1.

- **0-Simplices (Vértices):** Representan las uniones de tuberías, puntos de inyección de la bomba y reservorios. El espacio vectorial generado por estos vértices es el espacio de las 0-cadenas, $C_0$. Un elemento de $C_0$ podría representar, por ejemplo, la acumulación neta de masa en cada nodo.[8]
- **1-Simplices (Aristas):** Representan las tuberías y válvulas. El espacio vectorial generado es $C_1$. Un elemento de $C_1$ es una configuración de flujos a través de toda la red.

## 2.2 El Operador de Borde ($\partial$) y la Conservación

El operador central en esta formulación es el **Operador de Borde** (Boundary Operator), denotado como $\partial_k$. Para nuestro sistema de dimensión 1, nos interesa $\partial_1$, que mapea aristas a vértices.[9] Sea una arista $e_k$ que conecta el nodo $v_i$ hacia el nodo $v_j$ (orientación $i \to j$):

$$\partial_1(e_k) = v_j - v_i$$

Este operador es lineal. Su representación matricial es la **Matriz de Incidencia** $B$ (a veces denotada como $D$ o $\mathcal{I}$). En esta matriz, las filas corresponden a los nodos y las columnas a las aristas.

- $B_{ik} = -1$ si el nodo $i$ es el origen de la arista $k$.
- $B_{jk} = +1$ si el nodo $j$ es el destino de la arista $k$.
- $B_{mk} = 0$ en cualquier otro caso.[10]

### 2.2.1 Interpretación Homológica de la Ley de Corrientes de Kirchhoff (KCL)

La Ley de Corrientes de Kirchhoff establece que la suma algebraica de flujos en un nodo es cero (para nodos sin almacenamiento ni inyección externa). Si representamos el estado de flujo de la red como un vector $f$ (una 1-cadena), la operación de calcular la divergencia neta en todos los nodos es simplemente aplicar el operador de borde:

$$\partial_1 f = \mathbf{0}$$
En términos de álgebra homológica, los flujos físicamente válidos (en ausencia de fuentes externas) son aquellos que pertenecen al **Núcleo** (Kernel) del operador de borde:

$$f \in \ker(\partial_1)$$
Los elementos de este núcleo se denominan **1-ciclos** ($Z_1$). Esto revela una verdad profunda: en una red pasiva cerrada, cualquier flujo persistente debe formar bucles cerrados.[3]

- **Relevancia para la Refactorización:** El código flux_condenser.py original probablemente permitía la creación o destrucción arbitraria de "flujo". La nueva clase HydraulicGraph debe imponer la restricción topológica de que $\partial_1 f = s$, donde $s$ es el vector de fuentes externas (las bombas).

## 2.3 El Operador de Coborde ($\delta$) y el Gradiente

El concepto dual es el operador de coborde, $\delta_0$, que actúa sobre las 0-cocadenas (presiones en los nodos) para producir 1-cocadenas (diferencias de presión en las aristas).

$$(\delta_0 p)(e_{ij}) = p(v_j) - p(v_i)$$
Esto es idéntico al operador gradiente discreto ($\nabla$). Matricialmente, bajo las bases canónicas, el operador de coborde es la transpuesta de la matriz de incidencia:

$$\delta_0 \equiv B^T$$
La Ley de Voltajes de Kirchhoff (KVL) o la unicidad del potencial de presión implica que la caída de presión a lo largo de un lazo cerrado debe sumar cero. Esto se satisface automáticamente si las caídas de presión se derivan de un potencial nodal $p$ mediante el operador de coborde: $\Delta p = -\delta_0 p$.[11]

---

# 3. Formulación en Álgebra Lineal Computacional

La unificación de la topología algebraica y las leyes constitutivas físicas nos lleva al sistema de ecuaciones lineales que el código refactorizado debe resolver. Aquí es donde el "álgebra lineal" solicitada se manifiesta operativamente.

## 3.1 Derivación de la Ecuación de Poisson en Grafos

Buscamos encontrar el estado de presión $p$ (vector $N \times 1$) y el estado de flujo $f$ (vector $M \times 1$) dados los parámetros de la red.

1. **Ecuación de Continuidad (Topología):** La divergencia del flujo en los nodos es igual a la inyección de las fuentes.
   $$B f = s$$
   Donde $s$ es el vector de fuentes (inyecciones de la bomba).
2. **Ley Constitutiva (Física):** El flujo en una arista es proporcional a la diferencia de presión negativa (fluye de mayor a menor presión).
   $$f = W (-\Delta p_{arista}) = W (-B^T p)$$
   Donde $W$ es una matriz diagonal de tamaño $M \times M$ que contiene las conductancias hidráulicas ($W_{kk} = 1/R_{h,k}$) de cada tubería.
3. **Sistema Laplaciano:** Sustituyendo (2) en (1):
   $$B = s$$
   $$-(B W B^T) p = s$$
   Definimos la **Matriz Laplaciana Ponderada** del grafo como $L = B W B^T$. El sistema final a resolver es:
   $$L p = -s$$
   .[13]

## 3.2 Análisis de la Matriz Laplaciana ($L$)

La matriz $L$ encapsula toda la información sobre la conectividad y la resistencia física de la red pasiva.

- **Propiedades:** Es simétrica y semidefinida positiva (para conductancias positivas).
- **Singularidad:** En una red cerrada sin nodos de referencia, las filas de $L$ suman cero. Esto implica que $L$ es singular ($\det(L) = 0$). Físicamente, esto significa que existen infinitas soluciones para la presión $p$, todas diferenciadas por una constante aditiva (presión hidrostática base).
- **Manejo de Condiciones de Frontera (Dirichlet):** Para hacer el sistema soluble, debemos fijar la presión en al menos un nodo (el "Nodo Tierra" o Reservorio). Si el nodo $k$ es el reservorio con $P_k = 0$:
  1. Eliminamos la fila $k$ y la columna $k$ de la matriz $L$, obteniendo la matriz reducida $L_{red}$.
  2. Eliminamos el elemento $k$ del vector de fuentes $s$, obteniendo $s_{red}$.
  3. Resolvemos el sistema no singular: $L_{red} \cdot p_{red} = -s_{red}$.
  4. Reconstruimos el vector $p$ completo insertando el valor conocido $P_k$.[15]

---

# 4. Modelado Físico de la Bomba Hidráulica Lineal

La transformación del "Condensador de Flujo" a una "Bomba Lineal" requiere una definición precisa de qué tipo de máquina hidráulica estamos simulando.

### 4.1 La Bomba de Desplazamiento Positivo como Fuente de Corriente

Para cumplir con el requerimiento de linealidad y control de flujo preciso, modelamos la bomba como una **Bomba de Desplazamiento Positivo** (ej. bomba de pistón, de engranajes o peristáltica).

- **Comportamiento Físico:** Esta máquina desplaza un volumen fijo de fluido por ciclo, independientemente de la presión de descarga (hasta el límite de par del motor).
- **Analogía de Circuito:** Se comporta exactamente como una **Fuente de Corriente Ideal Independiente** en electrónica.[16] Inyecta una corriente $I_s$ fija en el circuito.
- **Modelado Matemático:** A diferencia de una tubería (que es una arista con peso en $L$), la bomba no es un elemento pasivo del Laplaciano. Es un elemento activo que modifica el vector $s$ (Right-Hand Side).
    - Si la bomba conecta el nodo $u$ (succión) al nodo $v$ (descarga) con caudal $Q_{bomba}$:
        - Restamos $Q_{bomba}$ del nodo $u$ ($s_u \leftarrow s_u - Q_{bomba}$).
        - Sumamos $Q_{bomba}$ al nodo $v$ ($s_v \leftarrow s_v + Q_{bomba}$).

### 4.2 Diferencias Críticas con la Lógica "Flux Condenser"

El código original flux_condenser.py, por su nombre, sugiere un dispositivo que "condensa" o almacena flujo (análogo a un capacitor eléctrico $C = dQ/dV$).

- **Corrección Conceptual:** Un capacitor almacena potencial (presión) acumulando carga (volumen). Una bomba *genera* flujo consumiendo trabajo externo.
- **Implicación en Refactorización:** Si el usuario desea mantener la noción de "condensador" como almacenamiento, se debe añadir un **Acumulador Hidráulico** al modelo. Sin embargo, dado que se solicita transformar la lógica *hacia* una bomba lineal, asumimos que la función de "condensador" (almacenamiento) se reemplaza o se reinterpreta como la función de "bombeo" (generación de flujo constante).[18]

---

## 5. Arquitectura de Software y Patrones de Diseño (Python 2025)

La implementación debe seguir los estándares modernos de ingeniería de software científico en Python, priorizando la claridad, la validación de tipos y la modularidad.

### 5.1 Estructura del Proyecto Refactorizado

Se propone la siguiente estructura de directorios, alineada con las mejores prácticas de empaquetado científico (estilo src, configuración pyproject.toml).[20]

```
linear_hydraulic_pump/
├── pyproject.toml # Configuración moderna de build y dependencias
├── src/
│ └── hydraulic_sim/
```

```
│   ├── __init__.py
│   ├── topology/
│   │   ├── __init__.py
│   │   └── graph_manager.py # Manejo de NetworkX y matrices B, L
│   ├── physics/
│   │   ├── __init__.py
│   │   ├── fluid.py # Propiedades (viscosidad, densidad)
│   │   └── components.py # Clases: Tubería, Bomba, Válvula
│   └── solver/
│   ├── __init__.py
│   └── linear_solver.py # Wrappers de SciPy Sparse
├── tests/
│   ├── test_conservation.py # Validación de KCL
│   └── test_pump_curve.py # Validación de comportamiento de fuente
└── examples/
└── simulate_circuit.py # Script demostrativo (reemplazo de flux_condenser.py)
```

## 5.2 Patrones de Diseño Orientado a Objetos (OOD)

Para gestionar la complejidad de la simulación, se aplicarán los siguientes patrones [22]:

1. **Patrón Composite (Compositor):** Para la red hidráulica. La clase HydraulicNetwork contendrá una colección de nodos y componentes (tuberías, bombas), permitiendo tratar la red como un objeto unificado que delega cálculos a sus partes.
2. **Patrón Strategy (Estrategia):** Para el cálculo de la conductancia. Las tuberías pueden tener diferentes modelos de fricción. Una interfaz ResistanceModel permitirá cambiar entre "Poiseuille" (lineal) y modelos futuros (turbulentos) sin modificar el núcleo del solver.
3. **Patrón Builder (Constructor):** Para facilitar la creación programática de redes complejas. NetworkBuilder.add_node(...).connect(...).

## 5.3 Implementación de Type Hinting y Validación

Siguiendo las prácticas de 2025, el código debe ser estrictamente tipado para permitir el análisis estático (mypy/pyright) y mejorar la robustez.[23]

Python

```
from typing import Protocol, List, Dict, Optional
import numpy.typing as npt
import numpy as np

class HydraulicComponent(Protocol):
    """Protocolo para cualquier componente conectable en la red."""
```

```
    def get_conductance(self) -> float:...

class LinearPump:
    """
    Modelo de Bomba de Desplazamiento Positivo (Fuente de Corriente).
    Reemplaza la lógica difusa de 'flux_condenser'.
    """
    def __init__(self, inlet_node: str, outlet_node: str, flow_rate: float) -> None:
        self.inlet: str = inlet_node
        self.outlet: str = outlet_node
        self.flow_rate: float = flow_rate  # m^3/s

    def inject_to_source_vector(self, s_vector: npt.NDArray[np.float64],
                                node_map: Dict[str, int]) -> None:
        """
        Aplica la divergencia de flujo al vector fuente global s.
        """
        if self.inlet in node_map:
            s_vector[node_map[self.inlet]] -= self.flow_rate
        if self.outlet in node_map:
            s_vector[node_map[self.outlet]] += self.flow_rate
```

---

# 6. Verificación, Validación y Estabilidad Numérica

La refactorización no está completa sin un marco de verificación que asegure que el código produce resultados físicamente posibles.

## 6.1 Teorema de Tellegen como Invariante de Prueba

El Teorema de Tellegen establece que en cualquier red de elementos concentrados, la suma de las potencias instantáneas es cero ($\sum v_k i_k = 0$). En nuestro contexto hidráulico:

$$\sum_{\text{componentes}} (\Delta P_k \cdot Q_k) = 0$$

Esto implica que la potencia suministrada por la bomba ($P_{bomba} = \Delta P \cdot Q$) debe ser exactamente igual a la potencia disipada por fricción viscosa en las tuberías. El código refactorizado debe incluir una aserción (assertion) que verifique esta igualdad dentro de una tolerancia numérica ($\epsilon < 10^{-9}$) después de cada solución.[25]

## 6.2 Estabilidad Espectral y Conectividad

Antes de intentar resolver $Lp = -s$, el sistema debe verificar la conectividad del grafo.

- **Número de Betti ($\beta_0$):** El número de componentes conexos del grafo. Para que

el sistema tenga solución única con un solo nodo de referencia, el grafo debe ser conexo ($\beta_0 = 1$).
- **Validación con NetworkX:** Se utilizará nx.is_connected(G) y nx.number_connected_components(G) para auditar la topología antes de ensamblar las matrices. Si el grafo está desconectado, la matriz Laplaciana será singular de una manera que la eliminación de un solo nodo tierra no resolverá para todos los subgrafos.[13]

---

# 7. Implementación de Referencia y Migración de Código

A continuación se detalla la lógica central del solver, demostrando la integración de las bibliotecas científicas para reemplazar el script original.

## 7.1 Construcción del Solver Laplaciano

El corazón de la nueva implementación reside en la clase HydraulicSolver. Esta clase orquesta la interacción entre la topología y el álgebra lineal.

Python

```
import networkx as nx
import scipy.sparse as sp
from scipy.sparse.linalg import spsolve

class HydraulicSolver:
    def solve_steady_state(self, network: 'HydraulicGraph') -> Dict[str, float]:
        # 1. Obtener Matriz Laplaciana Ponderada (L)
        # NetworkX retorna L = D - A, que es equivalente a B W B.T
        L_matrix = nx.laplacian_matrix(network.graph, weight='conductance')

        nodes = list(network.graph.nodes())
        node_map = {node: i for i, node in enumerate(nodes)}
        n = len(nodes)

        # 2. Construir Vector de Fuentes (s)
        # s representa la inyección neta de caudal en cada nodo
        s_vector = np.zeros(n)
        for pump in network.pumps:
            pump.inject_to_source_vector(s_vector, node_map)
```

```
    # 3. Aplicar Condiciones de Frontera (Dirichlet)
    # Se requiere al menos un nodo reservorio con P=0 para evitar singularidad
    if not network.reservoirs:
        raise ValueError("Topología inválida: Se requiere al menos un reservorio (tierra).")

    # Máscara para nodos libres (no reservorios)
    free_nodes_mask = np.ones(n, dtype=bool)
    for res_node in network.reservoirs:
        idx = node_map[res_node]
        free_nodes_mask[idx] = False
        # La ecuación para el nodo reservorio se elimina del sistema

    # 4. Reducción del Sistema
    # L_red * p_free = -s_red
    L_red = L_matrix[free_nodes_mask, :][:, free_nodes_mask]
    s_red = s_vector[free_nodes_mask]

    # Nota de Signo: Dependiendo de la definición de L (Grado - Adyacencia),
    # L * p representa el flujo SALIENTE neto debido a la presión.
    # Conservación: Flujo Saliente = Inyección Entrante.
    # L * p = s  => p = inv(L) * s.
    # (Se debe verificar el signo según la convención de NetworkX).

    # 5. Resolución Numérica Dispersa
    # Usamos spsolve para eficiencia O(N) en lugar de O(N^3)
    p_free = spsolve(L_red, s_red)

    # 6. Reconstrucción de Resultados
    pressures = np.zeros(n)
    pressures[free_nodes_mask] = p_free

    return {node: p for node, p in zip(nodes, pressures)}
```

## 7.2 Migración del 'Flux Condenser'

Para completar la refactorización, se deben mapear las funciones del archivo flux_condenser.py original a métodos de la nueva clase.

- Si flux_condenser.activate() iniciaba un bucle, ahora debe llamar a solver.solve_steady_state().
- Si flux_condenser.set_capacity() definía un límite, ahora debe traducirse a network.add_reservoir() (definiendo un nodo con capacidad infinita a presión constante).
- Si flux_condenser.connect() establecía rutas, ahora es network.add_pipe().

# Conclusión

La presente investigación técnica ha delineado la ruta crítica para transformar el código flux_condenser.py de una abstracción fenomenológica a un simulador de ingeniería hidráulica de alta fidelidad. Mediante la aplicación rigurosa de la **Topología Algebraica**, hemos garantizado que las leyes de conservación de masa y energía sean respetadas estructuralmente por la matriz Laplaciana del grafo. Mediante la **Física de Circuitos**, hemos establecido una analogía lineal válida para el régimen de flujo laminar (Poiseuille), y mediante el **Álgebra Lineal Computacional**, hemos definido un método de solución numéricamente estable y escalable.
La implementación de esta refactorización resultará en un módulo de software robusto, capaz de simular redes complejas con bombas lineales, proporcionando una base sólida para futuros desarrollos en control de fluidos y simulación de gemelos digitales.

### Obras citadas


1. Electric Circuits and the Hydraulic Analogy, fecha de acceso: enero 27, 2026, https://ataridogdaze.com/science/hydraulic/
2. Hydraulic analogy - Wikipedia, fecha de acceso: enero 27, 2026, https://en.wikipedia.org/wiki/Hydraulic_analogy
3. Homology, equilibrium, and conservation laws I - arXiv, fecha de acceso: enero 27, 2026, https://arxiv.org/pdf/1901.03171
4. i have a question about the analogy between Hydraulic system and Electrical system, fecha de acceso: enero 27, 2026, https://electronics.stackexchange.com/questions/268490/i-have-a-question-about-the-analogy-between-hydraulic-system-and-electrical-syst
5. Application of algebraic topology to graphs and networks - Oregon State University, fecha de acceso: enero 27, 2026, https://ir.library.oregonstate.edu/downloads/tb09j920v
6. Modeling Fluid Systems, fecha de acceso: enero 27, 2026, https://www.site.uottawa.ca/~rhabash/ESSModelFluid.pdf
7. [1901.03171] Homology, equilibrium, and conservation laws I: Discrete systems of points, fecha de acceso: enero 27, 2026, https://arxiv.org/abs/1901.03171
8. Computing Homology - Math ∩ Programming, fecha de acceso: enero 27, 2026, https://www.jeremykun.com/2013/04/10/computing-homology/
9. Topological Data Analysis with Persistent Homology - Raphaël Tinarrage, fecha de acceso: enero 27, 2026, https://raphaeltinarrage.github.io/files/EMAp/SummerCourseTDA.pdf
10. incidence_matrix — NetworkX 3.6.1 documentation, fecha de acceso: enero 27, 2026, https://networkx.org/documentation/stable/reference/generated/networkx.linalg.graphmatrix.incidence_matrix.html
11. Although topology was recognized by Gauss and Maxwell to play a pivotal role in the formulation of electromagnetic boundary valu - The Library at SLMath, fecha

de acceso: enero 27, 2026, https://library.slmath.org/books/Book48/files/gross-kotiuga.pdf
12. Topology and Elementary Electric Circuit Theory, I, fecha de acceso: enero 27, 2026, https://www.math.stonybrook.edu/~tony/whatsnew/oct18/circuits.html
13. laplacian_matrix — NetworkX 3.6.1 documentation, fecha de acceso: enero 27, 2026, https://networkx.org/documentation/stable/reference/generated/networkx.linalg.laplacianmatrix.laplacian_matrix.html
14. Algorithms, Graph Theory, and Linear Equa- tions in Laplacian Matrices - Department of Computer Science, fecha de acceso: enero 27, 2026, https://www.cs.yale.edu/homes/spielman/PAPERS/icm10post.pdf
15. Handling Dirichlet Boundary Conditions, fecha de acceso: enero 27, 2026, https://dakiefer.net/uploads/CAE%20von%20Sensoren%20und%20Aktoren/slides_dirichlet_node_handling.pdf
16. How can I model a positive displacement pump using Pipe Flow Expert?, fecha de acceso: enero 27, 2026, https://www.pipeflow.com/software-technical-support/pipe-flow-expert-positive-displacement-pumps
17. Positive Displacement Pump Selection - PDH Academy, fecha de acceso: enero 27, 2026, https://pdhacademy.com/wp-content/uploads/2023/08/448-Positive-Displacement-Pump-Selection.pdf
18. 3.3 Positive displacement pump theory - Pumpfocus, fecha de acceso: enero 27, 2026, https://pumpfocus.com/pumpbook/positive-displacement-pump-theory/
19. Reciprocating positive displacement pumps - LEWA, fecha de acceso: enero 27, 2026, https://www.lewa.com/fileadmin/5_competence/2_whitepapers/Whitepaper_Reciprocating_positive_displacement_pumps.pdf
20. How to Structure Python Projects - Dagster, fecha de acceso: enero 27, 2026, https://dagster.io/blog/python-project-best-practices
21. drivendataorg/cookiecutter-data-science: A logical, reasonably standardized, but flexible project structure for doing and sharing data science work. - GitHub, fecha de acceso: enero 27, 2026, https://github.com/drivendataorg/cookiecutter-data-science
22. Design Patterns in Python - Refactoring.Guru, fecha de acceso: enero 27, 2026, https://refactoring.guru/design-patterns/python
23. Writing Python like it's Rust - more robust code with type hints — Jakub Beránek - YouTube, fecha de acceso: enero 27, 2026, https://www.youtube.com/watch?v=OFRLKWacOoA
24. Python Typing Survey 2025: Code Quality and Flexibility As Top Reasons for Typing Adoption - Engineering at Meta, fecha de acceso: enero 27, 2026, https://engineering.fb.com/2025/12/22/developer-tools/python-typing-survey-2025-code-quality-flexibility-typing-adoption/
25. Matrices Applied to Electrical Circuits - Linear Algebra Applications S19 - WordPress.com, fecha de acceso: enero 27, 2026,

https://linearalgebraapplications19.wordpress.com/2019/03/30/matrices-applied-to-electrical-circuits/