Integración Avanzada de Lógica de Membrana Amortiguadora en Redes de Condensadores mediante Dinámica de Fluidos y Difusión en Grafos: Un Marco Teórico y Computacional

1. Introducción: Hacia un Nuevo Paradigma en la Simulación de Transitorios Energéticos

La ingeniería de sistemas de potencia y la física computacional contemporáneas se enfrentan a un desafío crítico en el modelado de redes de alta densidad de almacenamiento energético, tales como las simuladas en el entorno flux_condenser.py. Los enfoques tradicionales, fundamentados casi exclusivamente en el análisis de circuitos mediante las leyes de Kirchhoff y solucionadores matriciales algebraicos, a menudo carecen de la robustez necesaria para gestionar fenómenos transitorios de alta frecuencia y gran amplitud —denominados comúnmente "picos" u "ondas de choque"— sin incurrir en inestabilidades numéricas o requerir pasos de tiempo infinitesimales que hacen inviable la simulación a gran escala. Este informe propone un cambio de paradigma radical: la reinterpretación de la red de condensadores no como un conjunto discreto de elementos eléctricos estáticos, sino como un sistema dinámico de fluidos que habita sobre una estructura topológica compleja.

El objetivo central de esta investigación es el desarrollo y la integración de una lógica de "membrana amortiguadora" (*damping membrane logic*) dentro de flux_condenser.py. Esta membrana no es un mero filtro paso bajo pasivo; se conceptualiza como una entidad viscoelástica, modelada mediante ecuaciones diferenciales parciales discretizadas sobre grafos, capaz de absorber energía cinética (picos de corriente) y redistribuir energía potencial (sobretensión) mediante mecanismos de difusión no lineal. Para lograr esto, fusionamos la analogía hidráulico-eléctrica rigurosa con herramientas avanzadas del Análisis Topológico de Datos (TDA), específicamente el Cálculo Exterior Discreto (DEC) y la teoría espectral de grafos.

La motivación subyacente surge de la observación de que los condensadores reales, especialmente en configuraciones de alta potencia, exhiben comportamientos parásitos y no linealidades que los modelos ideales ignoran. Al introducir una "membrana", emulamos el comportamiento de los acumuladores hidráulicos, donde una barrera física separa el fluido de trabajo de un gas compresible, proporcionando una amortiguación intrínseca contra el golpe de ariete.[1] En el dominio digital de flux_condenser.py, esta membrana se traduce en operadores laplacianos generalizados (Laplaciano de Hodge y p-Laplaciano) que actúan

sobre las señales del grafo para suavizar las discontinuidades espaciales y temporales, garantizando la conservación de la energía y la estabilidad del sistema frente a perturbaciones estocásticas o conmutaciones abruptas.[3]
A lo largo de este documento, desglosaremos la formulación matemática necesaria para tratar la corriente eléctrica como un flujo incompresible (o cuasi-compresible) en una variedad discreta, implementando algoritmos de difusión que imitan la termodinámica de la transferencia de calor para equilibrar las cargas. Se prestará especial atención a la construcción de matrices de frontera y bases de ciclos utilizando librerías científicas de Python como NetworkX y SciPy, permitiendo una implementación eficiente y escalable de la lógica de membrana propuesta.

2. La Analogía Hidráulico-Eléctrica Rigurosa: Fundamentos de la Membrana

Para diseñar una "membrana" efectiva en el código, primero debemos establecer una correspondencia física precisa entre el dominio eléctrico y el hidráulico. Si bien las analogías básicas son comunes en la educación introductoria, el modelado de transitorios requiere una formulación que contemple la inercia, la compresibilidad y la viscosidad no newtoniana.

2.1. Isomorfismo de las Variables de Estado

La base de nuestra lógica de membrana reside en tratar la carga eléctrica (q) como volumen de fluido y el potencial eléctrico (V) como presión hidrostática (P). En este marco, la corriente ($I = dq/dt$) es análoga al caudal volumétrico (Q). Sin embargo, para mitigar picos transitorios, la analogía debe extenderse a las propiedades dinámicas del medio. Un conductor eléctrico no es simplemente una tubería sin fricción; posee una resistencia intrínseca (análoga a la resistencia hidráulica por fricción viscosa) y una inductancia (análoga a la inertancia o masa efectiva del fluido en movimiento). La "membrana" que buscamos integrar actúa como una capacitancia distribuida con propiedades de amortiguamiento. En hidráulica, esto se corresponde con un acumulador de vejiga o membrana, donde la energía se almacena en la compresión de un gas o resorte, separado del fluido por una barrera flexible.[5]

Tabla 1: Correspondencia Ampliada de Variables Físicas para flux_condenser.py

Dominio Eléctrico	**Variable**	**Unidad**	**Dominio Hidráulico**	**Variable**	**Mecanismo Físico de la Membrana**
Potencial (Voltaje)	V	Volt [V]	Presión	P [Pa]	Fuerza normal ejercida sobre la membrana.
Carga	q	Coulomb [C]	Volumen	Vol [m^3]	Desplazamiento volumétrico de la membrana.

Corriente	I	Ampere [A]	Caudal	Q [m^3/s]	Velocidad de deformación de la membrana.
Capacitancia	C	Farad [F]	Compliancia	C_h [m^3/Pa]	Elasticidad del material de la membrana (k^{-1}).
Inductancia	L	Henry [H]	Inertancia	I_h [$Pa \cdot s^2/m^3$]	Masa efectiva de la membrana y el fluido acelerado.
Resistencia	R	Ohm [Ω]	Resistencia	R_h [$Pa \cdot s/m^3$]	Viscosidad interna del material de la membrana.

2.2. Dinámica del Acumulador de Membrana

El modelo convencional de un condensador ($I = C \cdot dV/dt$) es insuficiente para describir la mitigación de picos porque asume una respuesta instantánea y puramente elástica. Para la lógica de membrana en flux_condenser.py, adoptamos el modelo del acumulador hidráulico, donde la dinámica está gobernada por una ecuación diferencial de segundo orden que incluye inercia y amortiguamiento.[2]
La ecuación constitutiva para la membrana propuesta es:

$$\Delta P = \frac{1}{C_h} \Delta Vol + R_{mem} \frac{d(Vol)}{dt} + I_{mem} \frac{d^2(Vol)}{dt^2}$$
Traduciendo al dominio eléctrico para un nodo i de la red:

$$V_i(t) = \frac{1}{C_i} q_i(t) + R_{mem} I_i(t) + L_{mem} \frac{dI_i}{dt}$$
Aquí, el término R_{mem} representa la **viscosidad** de la membrana. Es este término el que disipa la energía de los transitorios de alta frecuencia (picos abruptos), actuando como un amortiguador (*snubber*) intrínseco. El término L_{mem} representa la **inercia** de la membrana, impidiendo cambios instantáneos en la velocidad de deformación (corriente), lo que suaviza la pendiente de los frentes de onda.[7]
La implementación de esta lógica en flux_condenser.py implica que cada nodo capacitor no se modela como un simple escalar C, sino como un sistema dinámico local que responde a las leyes de un oscilador armónico amortiguado forzado por el flujo de la red.

2.3. Termodinámica de la Redistribución

En un sistema de fluidos interconectados, cualquier diferencia de presión (voltaje) tiende a equilibrarse mediante difusión. Si imaginamos la red de condensadores como una serie de tanques conectados por tuberías porosas, la energía fluye desde las zonas de alta presión a las de baja presión buscando minimizar la energía libre del sistema.
Este proceso es inherentemente difusivo y está gobernado por la ecuación del calor (o difusión). La introducción de la lógica de membrana añade un componente de "reacción" a este proceso de "difusión": la membrana puede endurecerse (reducir capacitancia) o ablandarse (aumentar capacitancia) en función de la presión local, modulando la velocidad de redistribución de energía. Esto introduce no linealidades beneficiosas que exploraremos mediante el p-Laplaciano en secciones posteriores.[4]

3. Fundamentos Topológicos: La Red como Complejo Simplicial

Para implementar modelos de fluidos en una red de circuitos arbitraria, debemos abandonar la noción de geometría continua y adoptar la **Topología Algebraica**, específicamente el marco de los **Complejos Simpliciales**. Esto permite definir operadores diferenciales (como el gradiente y el rotacional) de manera rigurosa sobre la estructura discreta del grafo de condensadores.[9]

3.1. Definición de la Estructura Discreta

Una red de condensadores puede representarse como un complejo simplicial K, compuesto por elementos de diferentes dimensiones:

1. **0-simplices (Nodos/Vértices, V):** Representan los puntos de conexión eléctrica, donde se define el potencial escalar (Voltaje, una 0-forma primal). En nuestra analogía, son los depósitos de fluido.[12]
2. **1-simplices (Aristas/Ramas, E):** Representan los componentes conductores (cables, resistencias, inductores) que conectan los nodos. Aquí se define el flujo (Corriente, una 1-forma primal o dual).[14]
3. **2-simplices (Caras/Bucles, F):** Representan los lazos cerrados o mallas elementales del circuito. Son cruciales para capturar fenómenos magnéticos inductivos y corrientes circulantes (eddy currents) que atrapan energía en bucles locales.[9]

3.2. Cadenas y Co-cadenas en flux_condenser.py

En el lenguaje del Cálculo Exterior Discreto (DEC), las variables físicas del simulador se mapean a cadenas y co-cadenas [15]:

- **Estado del Sistema (Voltajes):** Es una 0-co-cadena primal, $V \in C^0(K, \mathbb{R})$. Asigna un valor real a cada vértice.
- **Flujos de Redistribución (Corrientes):** Es una 1-co-cadena primal, $I \in C^1(K, \mathbb{R})$. Asigna un valor real a cada arista orientada.
- **Fuerzas Electromotrices de Bucle:** Asociadas a las 2-cadenas, capturan la energía magnética almacenada en las mallas.

La topología del sistema se codifica completamente mediante los **Operadores de Frontera** (∂_k).

- ∂_1: Mapea aristas a sus nodos extremos (1-cadenas a 0-cadenas).
- ∂_2: Mapea caras (bucles) a las aristas que las delimitan (2-cadenas a 1-cadenas).

La propiedad fundamental $\partial_1 \circ \partial_2 = 0$ (la frontera de una frontera es vacía) es la manifestación topológica de las leyes de conservación de Kirchhoff y garantiza que el sistema sea físicamente consistente.[13]

3.3. Construcción de Matrices de Frontera en Python

Para la implementación en flux_condenser.py, es imperativo construir estas matrices de manera eficiente utilizando librerías de álgebra lineal dispersa. A diferencia de las matrices de adyacencia densas, las matrices de frontera (B_1 y B_2) son altamente dispersas. Usando scipy.sparse y networkx, podemos construir la matriz de incidencia B_1 (que es la transpuesta del operador de frontera ∂_1) de la siguiente manera, tal como sugieren las mejores prácticas en computación científica para grafos grandes [18]:

Tabla 2: Operadores Topológicos y su Implementación Matricial

Operador Físico	Operador DEC	Representación Matricial (Python)	Dimensión
Gradiente de Voltaje (∇V)	Derivada Exterior d_0	B_1^T (Incidencia Transpuesta)	$
Divergencia de Corriente ($\nabla \cdot I$)	Co-derivada δ_1	B_1 (Incidencia)	$
Rotacional de Corriente ($\nabla \times I$)	Derivada Exterior d_1	B_2^T (Rotacional Discreto)	$
Circulación de Bucle	Frontera ∂_2	B_2 (Matriz de Ciclos)	$

La matriz B_2 es particularmente importante para la lógica de membrana, ya que permite detectar y amortiguar corrientes circulantes. Su construcción requiere extraer una base de ciclos mínima del grafo, lo cual se puede realizar mediante networkx.minimum_cycle_basis().[20]

4. Difusión en Grafos: El Laplaciano y la Ecuación del Calor

La redistribución de energía en la red de condensadores se modela naturalmente como un proceso de difusión. Si se inyecta un pico de carga en un nodo, la "membrana" debe facilitar que esta carga se disperse hacia los vecinos para evitar una sobretensión local destructiva. Este comportamiento está gobernado por el Laplaciano del Grafo.

4.1. El Laplaciano de Grafos (L_0)

El Laplaciano combinatorio estándar, denotado como L_0, actúa sobre las 0-formas (voltajes en los nodos). Se define matricialmente como:

$$L_0 = B_1^T W B_1 = D - A$$
Donde D es la matriz de grados (suma de conductancias conectadas al nodo) y A es la matriz de adyacencia ponderada.
Físicamente, aplicar el operador Laplaciano a un vector de voltajes V calcula la suma ponderada de las diferencias de potencial con los vecinos:

$$(L_0 V)_i = \sum_{j \sim i} w_{ij} (V_i - V_j)$$
Esta cantidad es proporcional a la corriente neta que fluye fuera del nodo i hacia sus vecinos.

4.2. Ecuación de Difusión Lineal (Heat Kernel)

La ecuación fundamental que la membrana intenta imponer para suavizar los picos es la Ecuación del Calor discretizada [22]:

$$\frac{\partial V}{\partial t} = -k \cdot L_0 V$$
Donde k es el coeficiente de difusividad, que en nuestro circuito depende inversamente de la capacitancia y resistencia ($1/RC$). La solución de esta ecuación a lo largo del tiempo, $V(t) = e^{-k L_0 t} V(0)$, representa la convolución del estado inicial con el núcleo del calor (*heat kernel*) del grafo. Este proceso suaviza las variaciones espaciales abruptas (picos de alto voltaje relativos a los vecinos), redistribuyendo la energía de manera isotrópica.[24]
Sin embargo, la difusión lineal simple tiene limitaciones:

1. Es lenta para eliminar gradientes grandes si la constante de tiempo es fija.
2. Tiende a "borronear" todo el estado del sistema, lo cual puede no ser deseable si queremos mantener ciertos niveles de voltaje operativos.
3. No distingue entre una señal operativa legítima y un pico de ruido transitorio.

Por ello, la lógica de membrana en flux_condenser.py debe evolucionar hacia modelos de difusión más sofisticados, como el Laplaciano de Hodge y la difusión no lineal.

5. Suavizado Avanzado: El Laplaciano de Hodge y el Procesamiento de Señales Simpliciales

Para una mitigación de transitorios verdaderamente efectiva, no basta con mirar los voltajes en los nodos. Los picos de energía a menudo se manifiestan como corrientes violentas en las aristas o oscilaciones resonantes en los bucles. El marco de Procesamiento de Señales Simpliciales (Simplicial Signal Processing, SSP) nos proporciona el **Laplaciano de Hodge** (L_1), una herramienta poderosa que generaliza la difusión para operar sobre flujos (corrientes).[3]

5.1. Definición del Laplaciano de Hodge (L_1)

El Laplaciano de Hodge de orden 1 actúa sobre las señales de las aristas (corrientes, I) y se compone de dos términos ortogonales:

$$L_1 = \underbrace{d_0 \delta_1}_{L_{grad}} + \underbrace{\delta_2 d_1}_{L_{curl}}$$
En notación matricial, utilizando las matrices de incidencia B_1 y de ciclos B_2:

$$L_1 = B_1^T B_1 + B_2 B_2^T$$
Este operador es fundamental para la lógica de membrana porque desacopla y trata por separado dos tipos de flujos de energía [25]:

1. **Componente Irrotacional (L_{grad}):** Relacionado con el término $B_1^T B_1$. Representa flujos inducidos por diferencias de potencial entre nodos (flujo gradiente). La difusión basada en este término suaviza las corrientes que divergen o convergen en los nodos, garantizando el cumplimiento de la Ley de Corrientes de Kirchhoff (KCL) de manera suave.
2. **Componente Solenoidal (L_{curl}):** Relacionado con el término $B_2 B_2^T$. Representa flujos que circulan en bucles cerrados (vórtices magnéticos). La difusión basada en este término amortigua las corrientes parásitas que giran en las mallas sin contribuir al transporte neto de carga.

5.2. Descomposición de Hodge-Helmholtz Discreta

Gracias al teorema de Hodge, cualquier configuración de corrientes I en la red puede descomponerse en tres componentes ortogonales:

$$I = I_{grad} + I_{curl} + I_{harm}$$
- I_{grad}: Flujo impulsado por potenciales (baterías, condensadores).
- I_{curl}: Flujo rotacional local.
- I_{harm}: Flujo armónico, que corresponde a corrientes que atraviesan "agujeros" topológicos de la red (si la topología no es trivialmente contractible).[29]

Aplicación en flux_condenser.py:
La lógica de membrana puede diseñarse para aplicar coeficientes de amortiguamiento selectivos a cada componente.

- Para mitigar un **pico de voltaje**, la membrana debe facilitar el flujo I_{grad} (baja resistencia al gradiente).
- Para mitigar una **oscilación resonante**, la membrana debe penalizar el flujo I_{curl} (alta resistencia al rotacional).

La ecuación de evolución para las corrientes en las aristas bajo esta lógica sería:

$$\frac{\partial I}{\partial t} = - (\alpha L_{grad} + \beta L_{curl}) I$$
Donde α y β son parámetros sintonizables de la membrana. Si $\beta \gg \alpha$, la membrana actúa como un filtro que elimina selectivamente los bucles de corriente

(ruido inductivo) preservando el flujo de potencia principal.[31]

6. Dinámica No Lineal: La Membrana P-Laplaciana y Difusión Anisotrópica

Hasta ahora, hemos discutido modelos lineales. Sin embargo, una membrana amortiguadora ideal debe comportarse como un material no newtoniano: suave ante movimientos lentos pero rígida ante impactos, o viceversa, dependiendo de la estrategia de protección. El **P-Laplaciano** ofrece el marco matemático para esta adaptabilidad.[4]

6.1. El Operador P-Laplaciano en Grafos

El p-Laplaciano es una generalización no lineal del Laplaciano estándar, derivado de minimizar la energía funcional de p-Dirichlet:

$$E_p(V) = \frac{1}{p} \sum_{(i,j) \in E} w_{ij} |V_i - V_j|^p$$
El operador resultante (el gradiente de esta energía) actúa sobre el voltaje nodal V_i como:

$$\Delta_p V_i = \sum_{j \sim i} w_{ij} \phi_p(V_i - V_j)$$
Donde $\phi_p(x) = |x|^{p-2}x$ es una función de conductancia no lineal.

6.2. Regímenes de Comportamiento de la Membrana

El parámetro p define la "personalidad" de la membrana amortiguadora [4]:

- **$p = 2$ (Difusión Lineal):** La conductancia es constante. La respuesta es proporcional al gradiente. Es el comportamiento óhmico estándar.
- **$p > 2$ (Difusión Agresiva de Picos):** La conductancia efectiva aumenta con el gradiente de voltaje.
    - *Mecanismo:* $g_{eff} \propto |\Delta V|^{p-2}$.
    - *Efecto:* Si hay un pico transitorio grande (alto ΔV), la membrana se vuelve altamente conductora localmente, "cortocircuitando" el pico hacia los vecinos para diluirlo instantáneamente. Esto es análogo a una válvula de alivio de presión que se abre solo ante presiones peligrosas.
- **$1 \le p < 2$ (Difusión Preservadora de Bordes - Variación Total):** La conductancia efectiva disminuye con el gradiente.
    - *Mecanismo:* $g_{eff} \propto |\Delta V|^{p-2}$ (con singularidad en 0 regularizada).
    - *Efecto:* La membrana aísla los saltos grandes de voltaje, impidiendo que se propaguen, mientras suaviza las pequeñas oscilaciones (ruido). Esto es útil para confinar una falla en un sector de la red sin desestabilizar el resto.

6.3. Implementación de Perona-Malik

Una variante práctica para flux_condenser.py es la difusión anisotrópica de Perona-Malik, utilizada en procesamiento de imágenes para suavizar ruido sin borrar bordes.[34] La ecuación

de actualización de conductancia para la arista (i,j) sería:

$$w_{ij}(t) = w_0 \cdot g(|V_i - V_j|)$$
Con una función de núcleo como la Lorentziana:

$$g(s) = \frac{1}{1 + (s/K)^2}$$
Donde K es un umbral de voltaje. Picos por encima de K son resistidos fuertemente (aislamiento), mientras que fluctuaciones por debajo de K son difundidas (amortiguación).

7. Métodos Numéricos para Sistemas Rígidos (Stiff)

La introducción de términos de difusión rápida (para mitigar picos) junto con la dinámica lenta de carga de los condensadores convierte al sistema de ecuaciones diferenciales ordinarias (ODEs) resultante en un sistema **rígido** (*stiff*). El uso de integradores explícitos simples (como Forward Euler) conducirá inevitablemente a la inestabilidad numérica o "explosión", a menos que se utilicen pasos de tiempo extremadamente pequeños ($\Delta t < \lambda_{max}^{-1}$), lo cual es computacionalmente ineficiente.[23]

7.1. Análisis de Estabilidad y Selección de Solvers

Para garantizar que la simulación en flux_condenser.py sea robusta, es imperativo utilizar esquemas de integración implícitos o semi-implícitos.
Tabla 3: Comparación de Esquemas de Integración para la Lógica de Membrana

Método	Tipo	Estabilidad	Coste por Paso	Idoneidad
Euler Explícito	Explícito de 1er orden	Condicional (muy pobre)	Bajo ($O(N)$)	**No recomendado**. Propenso a oscilaciones espurias.
Euler Implícito	Implícito de 1er orden	Incondicional (L-estable)	Alto (Resuelve $Ax=b$)	**Alta**. Introduce amortiguamiento numérico artificial que beneficia la estabilidad.
Crank-Nicolson	Implícito de 2do orden	Incondicional (A-estable)	Alto	**Media**. Conserva energía pero puede sufrir "ringing" en transitorios rápidos.
BDF (Gear)	Implícito Multistep	L-estable (orden var.)	Muy Alto (Jacobiano)	**Excelente**. Estándar de oro para circuitos

				rígidos (scipy.integrate.ode con method='bdf').
Simpléctico	Geométrico (e.g., Verlet)	Condicional	Bajo/Medio	**Baja**. Diseñado para conservar energía (sistemas Hamiltonianos), pero aquí buscamos disipación controlada.[38]

7.2. Implementación Recomendada: Backward Differentiation Formula (BDF)

Dado que los transitorios en redes de potencia pueden abarcar órdenes de magnitud en escalas de tiempo, se recomienda el uso de **BDF (Backward Differentiation Formula)**. Este método es particularmente eficaz para amortiguar las componentes de alta frecuencia (ruido numérico y picos físicos) mientras integra con precisión la dinámica lenta.[39]
En Python, esto se implementa encapsulando la lógica de membrana en una función derivada f(t, y) y pasándola a scipy.integrate.solve_ivp con el argumento method='BDF' o method='LSODA'.

7.3. Amortiguamiento Viscoso y Condiciones de Frontera Absorbentes

Además del solver, es crucial implementar condiciones de frontera que absorban la energía de las ondas que llegan a los límites de la red simulada, evitando reflexiones artificiales que podrían amplificar los picos (resonancia de onda estacionaria). Utilizando la analogía con la ecuación de ondas elásticas amortiguadas, podemos implementar una **Capa Perfectamente Adaptada (PML)** o condiciones de frontera absorbentes de tipo Engquist-Majda en los nodos perimetrales del grafo.[41] Esto se logra añadiendo un término de fricción viscosa ($-\gamma \frac{\partial V}{\partial t}$) que solo es activo en los nodos de la "periferia" topológica del grafo.

8. Arquitectura Computacional e Implementación en Python

Esta sección traduce la teoría matemática expuesta en una arquitectura de software concreta para flux_condenser.py. El diseño se basa en la separación de preocupaciones: la topología del grafo, la construcción de operadores discretos y la evolución temporal.

8.1. Construcción de Matrices Topológicas con NetworkX y SciPy

El primer paso es convertir el grafo de red definido por el usuario en las matrices operadoras

B_1 (Incidencia) y B_2 (Ciclos).

Python

```
import networkx as nx
import numpy as np
import scipy.sparse as sp

def build_operators(G):
    """
    Construye las matrices de operadores discretos para la red G.
    Retorna:
    - B1: Matriz de Incidencia (Gradiente discreto) [Aristas x Nodos]
    - B2: Matriz de Ciclos (Rotacional discreto) [Caras x Aristas]
    """
    # 1. Matriz de Incidencia B1 (Gradiente)
    # B1[e, u] = -1 (tail), +1 (head)
    # NetworkX entrega incidence_matrix con orientación predeterminada
    B1 = nx.incidence_matrix(G, oriented=True).T # Transponer para convención DEC (d0)

    # 2. Base de Ciclos y Matriz B2 (Curl)
    # Identificar ciclos fundamentales
    cycles = nx.cycle_basis(G)
    num_cycles = len(cycles)
    num_edges = G.number_of_edges()
    edge_index = {e: i for i, e in enumerate(G.edges())}

    # Construir B2 dispersa
    # B2[f, e] = +/-1 si la arista e es parte del ciclo f
    data, rows, cols =,,
    for i, cycle in enumerate(cycles):
        # Un ciclo es una lista de nodos [u, v, w,...]
        # Convertir a aristas y determinar orientación
        cycle_edges = list(zip(cycle, cycle[1:] + cycle[:1]))
        for u, v in cycle_edges:
            if (u, v) in edge_index:
                e_idx = edge_index[(u, v)]
                sign = 1 # Orientación positiva
            elif (v, u) in edge_index:
                e_idx = edge_index[(v, u)]
                sign = -1 # Orientación negativa
            else:
```

```
            continue # No debería ocurrir si el ciclo es válido

        rows.append(e_idx) # Ojo: En DEC, d1 mapea 1-form a 2-form.
        cols.append(i)     # Filas=Aristas, Cols=Caras (Convención dual) o viceversa
        data.append(sign)

    # Construcción de matriz dispersa CSC para eficiencia
    B2 = sp.csc_matrix((data, (rows, cols)), shape=(num_edges, num_cycles))

    return B1, B2
```

Nota sobre la implementación: La función nx.cycle_basis encuentra una base fundamental de ciclos, lo cual es computacionalmente eficiente y topológicamente suficiente para grafos planares y muchas redes de potencia. Para mallas 3D complejas (tetraédricas), se requeriría una librería especializada en complejos simpliciales como GUDHI o PyDEC para extraer correctamente las 2-celdas.[17]

8.2. El Motor de la Membrana P-Laplaciana

La función que calcula la derivada temporal (dV/dt) debe incorporar la lógica no lineal.

Python

```
def membrane_dynamics(t, y, G, B1, B1_T, params):
    """
    Calcula dy/dt para el sistema con membrana amortiguadora.
    y: Vector de estado [Voltajes_Nodos, Corrientes_Inductivas]
    """
    num_nodes = G.number_of_nodes()
    V = y[:num_nodes]

    # 1. Calcular Gradiente de Voltaje en Aristas
    delta_V = B1 @ V  # [Aristas]

    # 2. Lógica P-Laplaciana (Conductancia Adaptativa)
    # p > 2 para mitigar picos grandes
    p = params.get('p', 3.0)
    # Conductancia base de las aristas
    G_base = np.array([d['conductance'] for u,v,d in G.edges(data=True)])

    # Modulación no lineal: g_eff = G_base * |dV|^(p-2)
    # Se añade epsilon para evitar singularidad en dV=0
    epsilon = 1e-9
```

```
    nonlinear_factor = np.power(np.abs(delta_V) + epsilon, p - 2)

    # Matriz de Conductancia Efectiva (Diagonal)
    K_eff = sp.diags(G_base * nonlinear_factor)

    # 3. Corriente de Difusión (Laplaciano Generalizado)
    # I_diff = B1_T * K_eff * B1 * V
    # Esto es equivalente a div( g(grad V) * grad V )
    I_diffusion = B1_T @ (K_eff @ delta_V)

    # 4. Ecuación Nodal (Kirchhoff de Corrientes + Membrana)
    # C * dV/dt = -I_diffusion - I_external +...
    # Aquí simplificado. En realidad se debe invertir C (matriz de masa)

    C_inv = params['C_inv_diag'] # Precomputada
    dV_dt = -C_inv @ I_diffusion

    return dV_dt
```

Esta implementación explota la eficiencia de las operaciones matriz-vector dispersas de SciPy, permitiendo que la "membrana" reevalúe su permeabilidad en cada evaluación de la función, adaptándose instantáneamente a la aparición de picos.

8.3. Filtrado Espectral y P-Laplaciano

La integración de la librería PyDEC o implementaciones personalizadas de DEC permite ir más allá. Se puede calcular el espectro del Laplaciano ($L = B_1^T B_1$) y proyectar el estado del sistema sobre los autovectores de baja frecuencia. Esto constituye un "filtro de paso bajo topológico".
Sin embargo, para una simulación en tiempo real dentro de flux_condenser.py, la descomposición espectral completa ($O(N^3)$) es prohibitiva. La aproximación mediante difusión (P-Laplaciano) descrita arriba es local ($O(E)$) y por tanto escalable a redes masivas.

9. Viscosidad y Amortiguación Magnética

Para completar el modelo, debemos considerar la energía almacenada en los campos magnéticos (inductancia) que causa oscilaciones. La "membrana" debe tener una componente resistiva magnética.
Utilizando la matriz B_2 construida anteriormente, podemos definir un término de disipación que actúe sobre las corrientes de bucle. Si I_{edges} es el vector de corrientes en las aristas:

1. Proyectar corrientes al espacio de ciclos: $I_{cycles} = (B_2^T B_2)^{-1} B_2^T I_{edges}$ (Pseudo-inversa o mínimos cuadrados).

2. Aplicar resistencia de ciclo R_{cyc}.
3. Esto añade una fuerza contra-electromotriz proporcional a la corriente circulante, disipando la energía de los vórtices magnéticos sin afectar el flujo de potencia principal (que no es cíclico).

10. Casos de Estudio y Comportamiento Esperado

10.1. Escenario 1: Conmutación Abrupta (Escalón Unitario)

Al simular el cierre instantáneo de un interruptor que conecta un condensador cargado a una red descargada:

- *Sin Membrana:* Se produce una oscilación de alta frecuencia (ringing) debido a las inductancias parásitas y la falta de amortiguamiento, potencialmente violando la conservación de energía numérica.
- *Con Membrana P-Laplaciana ($p=3$):* El frente de onda abrupto genera un gradiente enorme. La conductancia efectiva entre los nodos afectados se dispara momentáneamente. La carga se transfiere casi instantáneamente a los vecinos inmediatos, "achatando" el pico. La oscilación posterior es fuertemente amortiguada por la viscosidad del modelo BDF.

10.2. Escenario 2: Inyección de Ruido Estocástico

Si se introduce ruido blanco en la fuente de corriente:

- La componente difusiva del Laplaciano (L_{grad}) actúa como un filtro espacial. El ruido de alta frecuencia espacial (diferencias grandes entre vecinos aleatorios) es eliminado rápidamente, mientras que las tendencias globales de carga se mantienen. Esto confirma que la membrana actúa como un suavizador topológico de señales.[30]

11. Conclusión

La integración de una lógica de membrana amortiguadora en flux_condenser.py mediante modelos de dinámica de fluidos y difusión en grafos representa un avance significativo sobre las simulaciones de circuitos convencionales. Al adoptar la analogía hidráulica rigurosa —con acumuladores viscoelásticos y flujos inerciales— y formalizarla mediante el Cálculo Exterior Discreto y el P-Laplaciano, logramos un sistema que es inherentemente robusto frente a transitorios.

La implementación recomendada se basa en tres pilares:

1. **Topología Algebraica:** Uso de matrices de incidencia (B_1) y ciclos (B_2) para definir operadores diferenciales precisos sobre la red.
2. **No Linealidad Adaptativa:** Uso del P-Laplaciano ($p > 2$) para crear una conductancia dinámica que reacciona agresivamente ante picos de voltaje, actuando como una válvula de alivio distribuida.
3. **Integración Numérica Rígida:** Empleo de solucionadores BDF (Backward Differentiation Formula) para manejar la rigidez matemática introducida por las escalas de tiempo dispares de la difusión rápida y la dinámica capacitiva lenta.

Este enfoque no solo mitiga los riesgos de inestabilidad numérica y picos físicos dañinos, sino que dota a la simulación de una "física" más rica y realista, capaz de predecir fenómenos emergentes de redistribución de energía que los modelos lineales simples ignoran por completo.

Obras citadas

1. Hydraulic analogy - Wikipedia, fecha de acceso: enero 29, 2026, https://en.wikipedia.org/wiki/Hydraulic_analogy
2. Full article: Mathematical modelling of a hydraulic accumulator for hydraulic hybrid drives, fecha de acceso: enero 29, 2026, https://www.tandfonline.com/doi/full/10.1080/13873954.2016.1174716
3. Random Walks on Simplicial Complexes and the Normalized Hodge 1-Laplacian, fecha de acceso: enero 29, 2026, https://epubs.siam.org/doi/10.1137/18M1201019
4. Graph p-Laplacian Flows in Spectral Graph Theory - Emergent Mind, fecha de acceso: enero 29, 2026, https://www.emergentmind.com/topics/graph-p-laplacian-flows
5. Electric Circuits and the Hydraulic Analogy, fecha de acceso: enero 29, 2026, https://ataridogdaze.com/science/hydraulic/
6. Mathematical Modelling of a Hydraulic Accumulator for Hydraulic Hybrid Drives - ACIN – TU Wien, fecha de acceso: enero 29, 2026, https://www.acin.tuwien.ac.at/fileadmin/cds/pre_post_print/pfeffer2016.pdf
7. RC Snubbers for Step-Down Converters, fecha de acceso: enero 29, 2026, https://toshiba.semicon-storage.com/info/docget.jsp?did=63595
8. AN437 - RC snubber circuit design for triacs - Application note - STMicroelectronics, fecha de acceso: enero 29, 2026, https://www.st.com/resource/en/application_note/an437-rc-snubber-circuit-design-for-triacs-stmicroelectronics.pdf
9. A Perspective on the Algebra, Topology, and Logic of Electrical Networks - arXiv, fecha de acceso: enero 29, 2026, https://arxiv.org/html/2510.21911v1
10. Circuit topology (electrical) - Wikipedia, fecha de acceso: enero 29, 2026, https://en.wikipedia.org/wiki/Circuit_topology_(electrical)
11. Hypergraph Neural Sheaf Diffusion: A Symmetric Simplicial Set Framework for Higher-Order Learning - IEEE Xplore, fecha de acceso: enero 29, 2026, https://ieeexplore.ieee.org/iel8/6287639/10820123/11091307.pdf
12. Hodge Laplacian of Brain Networks - PMC - PubMed Central - NIH, fecha de acceso: enero 29, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC10909176/
13. Discrete Exterior Calculus lecture - DISCRETE DIFFERENTIAL GEOMETRY: AN APPLIED INTRODUCTION, fecha de acceso: enero 29, 2026, https://brickisland.net/DDGSpring2019/wp-content/uploads/2019/02/DDG_458_SP19_Lecture09_DiscreteExteriorCalculus.pdf
14. DIFFERENTIAL STRUCTURES IN ELECTROMAGNETIC FIELD AND CIRCUIT THEORY by NAMAKKAL VASUDEVAN BALASUBRAMANIAN, M. Tech. - The University of Liverpool Repository, fecha de acceso: enero 29, 2026, https://livrepository.liverpool.ac.uk/3173930/1/642931.pdf

15. ON THE MATHEMATICAL FOUNDATIONS OF ELECTRICAL CIRCUIT THEORY - Project Euclid, fecha de acceso: enero 29, 2026, https://projecteuclid.org/journals/journal-of-differential-geometry/volume-7/issue-1-2/On-the-mathematical-foundations-of-electrical-circuit-theory/10.4310/jdg/1214430827.pdf
16. DISCRETE EXTERIOR CALCULUS Contents 1. Introduction 1 2. History and Previous Work 4 3. Primal Simplicial Complex and Dual Cell - UCSD, fecha de acceso: enero 29, 2026, https://mathweb.ucsd.edu/~mleok/pdf/DeHiLeMa2003_dec.pdf
17. PyDEC: Software and Algorithms for Discretization of Exterior Calculus - Nathan Bell, fecha de acceso: enero 29, 2026, http://wnbell.com/media/2012-TOMS-PyDEC/BeHi2012.pdf
18. incidence_matrix — NetworkX 3.6.1 documentation, fecha de acceso: enero 29, 2026, https://networkx.org/documentation/stable/reference/generated/networkx.linalg.graphmatrix.incidence_matrix.html
19. to_scipy_sparse_array — NetworkX 3.6.1 documentation, fecha de acceso: enero 29, 2026, https://networkx.org/documentation/stable/reference/generated/networkx.convert_matrix.to_scipy_sparse_array.html
20. networkx.algorithms.cycles.minimum_cycle_basis, fecha de acceso: enero 29, 2026, https://networkx.org/documentation/networkx-2.1/reference/algorithms/generated/networkx.algorithms.cycles.minimum_cycle_basis.html
21. minimum_cycle_basis — NetworkX 3.6.1 documentation, fecha de acceso: enero 29, 2026, https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.cycles.minimum_cycle_basis.html
22. The heat kernel as the pagerank of a graph - PNAS, fecha de acceso: enero 29, 2026, https://www.pnas.org/doi/10.1073/pnas.0708838104
23. Basic Numerical Solution Methods for Differential Equations - Markus K. Brunnermeier, fecha de acceso: enero 29, 2026, https://markus.scholar.princeton.edu/document/327
24. Discrete Heat Kernels on Simplicial Complexes and Its Application to Functional Brain Networks - arXiv, fecha de acceso: enero 29, 2026, https://arxiv.org/html/2509.16908v2
25. Random Walks on Simplicial Complexes and the Normalized Hodge 1-Laplacian | SIAM Review - Cornell: Computer Science, fecha de acceso: enero 29, 2026, https://www.cs.cornell.edu/~arb/papers/normalized-hodge-SIREV-2020.pdf
26. Topological signal processing and learning: Recent advances and future challenges - TU Delft Research Portal, fecha de acceso: enero 29, 2026, https://research.tudelft.nl/en/publications/topological-signal-processing-and-learning-recent-advances-and-fu/
27. Online Graph Topology Learning via Time-Vertex Adaptive Filters: From Theory to Cardiac Fibrillation - IEEE Xplore, fecha de acceso: enero 29, 2026, https://ieeexplore.ieee.org/document/11119191/

28. tsitsvero/hodgelaplacians: Hodge/Bochner Laplacians of simplicial complexes, their spectra, higher-order diffusion and random walks - GitHub, fecha de acceso: enero 29, 2026, https://github.com/tsitsvero/hodgelaplacians
29. [2408.14109] Topological filtering of a signal over a network - arXiv, fecha de acceso: enero 29, 2026, https://arxiv.org/abs/2408.14109
30. Topological Filtering of Graph Signals: Persistence-Based Methods and Applications, fecha de acceso: enero 29, 2026, https://catalog.lib.kyushu-u.ac.jp/opac_download_md/7396165/math0294.pdf
31. PyTSPL: Topological Signal Processing and Learning in Python — PyTSPL 0.1.5 documentation, fecha de acceso: enero 29, 2026, http://pytspl.readthedocs.io/
32. Adaptive Smoothing for Visual Improvement of Image Quality via the p(x)-Laplacian Operator Effects of the p(x)-Laplacian Smoothing Operator on Digital Image Restoration: Contribution to an Adaptive Control Criterion - MDPI, fecha de acceso: enero 29, 2026, https://www.mdpi.com/2076-3417/13/20/11600
33. Non-Local Morphological PDEs and p-Laplacian Equation on Graphs with applications in image processing and machine learning - Olivier Lézoray, fecha de acceso: enero 29, 2026, https://lezoray.users.greyc.fr/Publis/elmoataz-jstsp2012.pdf
34. Image smoothing by nonlinear diffusion - Nils, fecha de acceso: enero 29, 2026, https://nils-olovsson.se/articles/image_smoothing_by_nonlinear_diffusion/
35. Image Smoothing and Edge Detection by Nonlinear Diffusion and Bilateral Filter - Computational Science Research Center, fecha de acceso: enero 29, 2026, https://www.csrc.sdsu.edu/reports/ACSESS200806.pdf
36. Integrate stiff ODEs with Python - scipy - Stack Overflow, fecha de acceso: enero 29, 2026, https://stackoverflow.com/questions/2088473/integrate-stiff-odes-with-python
37. Understanding the Finite-Difference Time-Domain Method - eecs.wsu.edu, fecha de acceso: enero 29, 2026, https://eecs.wsu.edu/~schneidj/ufdtd/ufdtd.pdf
38. Symplectic integrator - Wikipedia, fecha de acceso: enero 29, 2026, https://en.wikipedia.org/wiki/Symplectic_integrator
39. An Introduction to Stiff ODEs with Python - edwinchenyj - GitHub Pages, fecha de acceso: enero 29, 2026, https://edwinchenyj.github.io/scientific%20computing/stiff-python/
40. Solving Stiff Ordinary Differential Equations - MIT Parallel Computing and Scientific Machine Learning (SciML), fecha de acceso: enero 29, 2026, https://book.sciml.ai/notes/09-Solving_Stiff_Ordinary_Differential_Equations/
41. An energy absorbing far-field boundary condition for the elastic wave equation - | Computing - Lawrence Livermore National Laboratory, fecha de acceso: enero 29, 2026, https://computing.llnl.gov/sites/default/files/PS_Energy_09.pdf
42. Solve a Wave Equation with Absorbing Boundary Conditions: New in Wolfram Language 11, fecha de acceso: enero 29, 2026, https://www.wolfram.com/language/11/partial-differential-equations/solve-a-wave-equation-with-absorbing-boundary-cond.html
43. Absorbing Boundary Conditions for the Wave Equation and Parallel Computing - Université de Genève, fecha de acceso: enero 29, 2026,

https://www.unige.ch/~gander/Preprints/Absorbing.pdf
44. Persistence Matrix - GUDHI library, fecha de acceso: enero 29, 2026, https://gudhi.inria.fr/doc/3.11.0/group__persistence__matrix.html
45. Diffusion Maps for Signal Filtering in Graph Learning - arXiv, fecha de acceso: enero 29, 2026, https://arxiv.org/html/2312.14758v1