Proyecto 2: Comparación de algoritmos para la solución de sistemas de ecuaciones lineales

December 16, 2025

Abstract

1 Algoritmos Clásicos para la Solución de Sistemas de Ecuaciones Lineales

La solución de sistemas lineales $A\mathbf{x} = \mathbf{b}$ es un problema fundamental en matemáticas aplicadas, computación científica e ingeniería. En esta sección se presenta un resumen de los algoritmos más utilizados, junto con sus ventajas y desventajas.

1.1 Métodos Directos

1.1.1 Eliminación Gaussiana (LU)

El método de eliminación Gaussiana factoriza la matriz como

$$A = LU,$$

donde L es triangular inferior y U triangular superior. Luego el sistema se resuelve mediante sustituciones hacia adelante y hacia atrás.

Ventajas

- Método robusto y bien entendido.
- Tiempo de cómputo determinista: $O(n^3)$.
- Adecuado para múltiples vectores $\mathbf{b}$ (factorización única).

Desventajas

- Costoso en memoria y cómputo para matrices muy grandes.
- Sensible a mal condicionamiento sin pivoteo.
- Difícil de paralelizar eficientemente en arquitecturas distribuidas sin técnicas avanzadas.

1.1.2 Descomposición de Cholesky

Para matrices simétricas definidas positivas, se utiliza

$$A = LL^T.$$

Ventajas

- Menor costo computacional: $\frac{1}{3}n^3$.
- Estabilidad numérica superior para matrices SPD.

Desventajas

- Sólo aplica a matrices SPD.
- Puede fallar si la matriz no cumple la condición.

1.1.3 Métodos Basados en Descomposición QR

$$A = QR,$$

con Q ortogonal y R triangular.

Ventajas

- Muy estable numéricamente.
- Adecuado para problemas mal condicionados o con mínimos cuadrados.

Desventajas

- Más costoso que LU: alrededor de $2n^3/3$.

1.2 Métodos Iterativos

1.2.1 Método de Jacobi

Basado en la descomposición

$$A = D + L + U,$$

la iteración es

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - (L + U)\mathbf{x}^{(k)}).$$

Ventajas

- Fácil de implementar.
- Altamente paralelizable (actualizaciones independientes).

Desventajas

- Convergencia lenta.
- Requiere que A sea diagonalmente dominante o SPD para asegurar convergencia.

1.2.2 Método de Gauss–Seidel

$$\mathbf{x}^{(k+1)} = (D + L)^{-1}(\mathbf{b} - U\mathbf{x}^{(k)}).$$

Ventajas

- Convergencia más rápida que Jacobi.
- Fácil de implementar.

Desventajas

- Difícil de paralelizar.
- Requiere condiciones similares a Jacobi para asegurar convergencia.

1.2.3 Método SOR

El método de Sobrerrelajación Sucesiva (Successive Over-Relaxation) introduce un parámetro ω:

$$\mathbf{x}^{(k+1)} = (D + \omega L)^{-1} \left[(1 - \omega)D - \omega U\right] \mathbf{x}^{(k)} + \omega(D + \omega L)^{-1}\mathbf{b}.$$

Ventajas

- Más rápido que Gauss–Seidel para un ω óptimo.

Desventajas

- Elegir ω es difícil.
- No siempre eficiente en matrices mal condicionadas.

1.2.4 Métodos de Krylov (CG, GMRES, BiCGstab)

Métodos modernos basados en subespacios de Krylov:

$$\mathcal{K}_k(A, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \dots\}.$$

Ventajas

- Altamente eficientes para matrices dispersas.
- Convergencia rápida con buenos precondicionadores.
- Método de Gradientes Conjugados (CG) es óptimo para SPD.

Desventajas

- Requieren precondicionamiento para un rendimiento óptimo.
- GMRES puede requerir gran memoria (reinicios necesarios).

1.3 Resumen de Elección de Método

- **Matriz densa y tamaño moderado**: LU, QR.
- **Matriz SPD**: Cholesky, CG.
- **Matriz dispersa grande**: Métodos de Krylov con precondicionador.
- **Arquitecturas paralelas masivas**: Jacobi (simple), CG/GMRES (eficientes).

2 Métodos Multigrid y Bibliotecas Modernas (PyAMG y NVIDIA AmgX)

Los métodos Multigrid (MG) constituyen una de las familias más eficientes para resolver sistemas lineales dispersos provenientes de discretizaciones de ecuaciones diferenciales parciales (EDPs), especialmente elípticas. Su eficiencia radica en combinar relajaciones locales con correcciones en mallas más gruesas, eliminando componentes de error tanto de alta como de baja frecuencia.

2.1 Idea General del Método Multigrid

Sea el sistema lineal

$$A_h \mathbf{x}_h = \mathbf{b}_h,$$

donde h denota la resolución de la malla. Los métodos de relajación como Jacobi o Gauss–Seidel reducen eficazmente los componentes de error de alta frecuencia, pero son lentos con las componentes de baja frecuencia. Multigrid compensa esta deficiencia transfiriendo el problema a una malla más gruesa donde dichas componentes se vuelven más oscilatorias y, por lo tanto, más fáciles de eliminar.

Esquema Típico Multigrid (V–cycle)

Un ciclo V estándar consta de los siguientes pasos:

1. **Suavizado (relajación)**: aplicar k_1 iteraciones de un método relajante:

$$\mathbf{x}_h^{(m+1)} = S_h(\mathbf{x}_h^{(m)}, \mathbf{b}_h).$$

2. **Cálculo del residuo**:

$$\mathbf{r}_h = \mathbf{b}_h - A_h \mathbf{x}_h.$$

3. **Restricción del residuo** a la malla gruesa:

$$\mathbf{r}_{2h} = R_{2h}^{h}\mathbf{r}_h.$$

4. **Solución aproximada del sistema en malla gruesa**:

$$A_{2h}\mathbf{e}_{2h} = \mathbf{r}_{2h}.$$

5. **Prolongación de la corrección**:

$$\mathbf{x}_h \leftarrow \mathbf{x}_h + P_h^{2h}\mathbf{e}_{2h}.$$

6. **Suavizado post–corrección**: aplicar k_2 iteraciones adicionales del suavizador.

Ventajas del Método Multigrid

- Complejidad óptima: $O(N)$ para N grados de libertad.
- Escalabilidad sobresaliente en problemas de gran tamaño.
- Eficiente para ecuaciones elípticas, difusivas y problemas Poisson–like.

Desventajas

- Requiere adecuada construcción de operadores de transferencia y suavizadores.
- Difícil de aplicar directamente a matrices no estructuradas sin heurísticas.
- Implementación más compleja que métodos de Krylov tradicionales.

2.2 Algebraic Multigrid (AMG)

Mientras que Multigrid clásico requiere jerarquías de mallas geométricas, el método AMG construye dichas mallas *automáticamente* a partir de la matriz A. AMG permite aplicar Multigrid a problemas complejos, mallas no estructuradas y coeficientes heterogéneos.

Ventajas del AMG

- No necesita malla geométrica (construcción puramente algebraica).
- Excelente como precondicionador para métodos de Krylov (GMRES, CG, BiCGstab).
- Adecuado para matrices muy grandes y dispersas.

Desventajas del AMG

- Mayor costo de precomputación que Multigrid geométrico.
- Sensible a la estructura del grafo de A.
- Configuración óptima depende del problema (coarsening, smoothing).

2.3 PyAMG: Implementación de AMG en Python

PyAMG es una biblioteca de Python que implementa una colección de métodos Multigrid algebraicos modernos. Está diseñada para integrarse con `NumPy`, `SciPy` y otras herramientas del ecosistema científico de Python.

Características principales

- Implementación completa de AMG: classical Ruge–Stüben, smoothed aggregation, bootstrap AMG.
- Soporte para matrices dispersas de `SciPy`.
- Fácil integración con `scipy.sparse.linalg` como precondicionador.
- Totalmente en CPU; ideal para prototipos y experimentación.

Ventajas

- Fácil de usar y de integrar con Python científico.
- Ideal para investigación y pruebas rápidas.
- Bien documentado y extensible.

Desventajas

- No acelera en GPU.
- Rendimiento limitado frente a bibliotecas en C++ o CUDA.

2.4 NVIDIA AmgX: AMG en GPU

NVIDIA AmgX es una biblioteca de alto rendimiento diseñada para resolver sistemas lineales dispersos mediante AMG y métodos de Krylov acelerados específicamente en GPU. Está escrita sobre CUDA y pensada para clústeres con múltiples GPUs.

Características principales

- Implementación altamente optimizada de AMG y precondicionadores híbridos.
- Métodos Krylov acelerados en GPU: CG, GMRES, BiCGstab.
- Soporte multi–GPU con escalamiento distribuido.
- Interfaz en C++, Python (mediante bindings) y MPI.
- Diseñado para aplicaciones CFD, Poisson, ecuaciones elípticas y FEM grandes.

Ventajas

- Rendimiento muy superior gracias a CUDA y paralelismo masivo.
- Ideal para simulaciones de gran escala.
- Excelente como precondicionador de Krylov en GPU.
- Puede integrarse en pipelines HPC con MPI + múltiples GPUs.

Desventajas

- Requiere hardware NVIDIA.
- Curva de aprendizaje considerable.
- Configuración compleja para problemas no estándar.

2.5 Resumen Comparativo

- **PyAMG**: adecuado para investigación, prototipos, sistemas medianos y entornos CPU en Python.
- **AmgX**: orientado a producción HPC, sistemas gigantes, cómputo en GPU y aplicaciones industriales.
- **AMG**: precondicionador de referencia para métodos iterativos modernos.
- **Multigrid geométrico**: mejor opción cuando la malla es regular y se conoce la estructura del problema.

3 Algoritmos Multinivel en GPU para la Solución de Ecuaciones Lineales

Un enfoque relevante dentro del estudio de algoritmos eficientes para la solución de sistemas lineales es el método multinivel optimizado para GPU presentado en *A GPU-based multi-level algorithm for boundary value problems* por Julián T Becerra-Sagredo. Este trabajo introduce un algoritmo denominado *Geometric Single-Grid Multi-Level* (GSGML), diseñado para resolver problemas de valor de frontera que, tras discretización, conducen a sistemas lineales de gran tamaño.

El objetivo principal del método es explotar el paralelismo masivo de las unidades de procesamiento gráfico (GPU) mediante una estrategia multinivel que evita la construcción explícita de jerarquías de mallas. A diferencia de los métodos multigrid clásicos, que requieren múltiples rejillas y operaciones de restricción/prolongación, el enfoque GSGML opera dentro de una única malla, aplicando operadores suavizadores y correcciones de manera jerárquica sobre la misma estructura discretizada.

Desde el punto de vista algorítmico, este enfoque reduce considerablemente el costo de comunicación y las operaciones de acceso a memoria, dos factores críticos en arquitecturas GPU. Además, el método mantiene estabilidad numérica y una convergencia comparable a los esquemas multigrid tradicionales, pero con una implementación más amigable para hardware altamente paralelo.

Los resultados reportados muestran una aceleración significativa frente a métodos clásicos ejecutados en CPU, y también frente a interpretaciones más directas de multigrid en GPU. La combinación de simplicidad geométrica, operación en una sola malla y adecuación natural al paralelismo hacen del GSGML un candidato importante en el diseño moderno de solvers lineales eficientes para problemas elípticos.

En conclusión, este algoritmo representa una contribución dentro del ecosistema de métodos iterativos acelerados por GPU, demostrando que la arquitectura del hardware puede y debe motivar nuevas formulaciones matemáticas para la solución eficiente de sistemas lineales a gran escala.

4 Desempeño de algoritmos para la solución de sistemas lineales

A continuación se presentan tablas con el desempeño de diferentes implementaciones de algoritmos para la solución de sistemas lineales en una laptop Pavilion HP con Geforce GTX 1650.

Los algoritmos para matrices llenas son (1) LU en python, (2) LU de numpy, (3) LU de cupy, (4) GMRES propio en C, (5) GMRES de scipy, (6) GMRES de cupy y (7) pyAMG.

(poner tabla para matrices llenas aquí, hacer comentarios sobre lo que se observa)

Los algoritmos para matrices de llena a esparsa son (1) GMRES de scipy y (2) GMRES de cupy. La densidad de las matrices son $d = 0.1$ y $d = 0.01$.

(poner tabla para matrices de llenas a esparsas aquí, hacer comentarios sobre lo que se observa)

Los algoritmos para matrices esparsas pentadiagonales son (1) ML-Sagredo-cuda, (2) ML-Sagredo-python-numba-cuda, (3) pyAMG para CPU.

(poner tabla para matrices esparsas pentadiagonales aquí, hacer comentarios sobre lo que se observa)

5 Estudio de la difusión multi-nivel

Como vimos en la sección anterior, los algoritmos más rápidos son lo que tienen múltiples mallas (multi-grid) o múltiples niveles (multi-level) y van resolviendo de los niveles gruesos a los finos.

A continuación presentamos figuras que ilustran el funcionamiento básico del método multi-malla.

(fórmula del cartón de huevos u = sin(4*pi*x)*sin(4*pi*y))

(ecuación de calor en 2D)

(figura 1, cartón de huevos iniial (Figure$_0$.*png*))

Cuando se corre diez pasos en una malla de 128×128, casi no produce difusión.

(figura 2, cartón de huevos final después de 10 pasos en malla 128x128 (Figure$_1$.*png*))

Cuando se corre diez pasos en una malla de 64×64, produce más difusión, reduciendo el error más rápido.

(figura 3, cartón de huevos final después de 10 pasos en malla 64x64 (Figure$_2$.*png*))

En una malla de 32×32 es notable la difusión.

(figura 4, cartón de huevos final después de 10 pasos en malla 32x32 (Figure$_3$.*png*))

En una malla de 16×16 la difusión lleva a la función casi a cero.

(figura 5, cartón de huevos final después de 10 pasos en malla 16x16 (Figure$_4$.*png*))

6 Conclusiones

Escriba sus conclusiones.