Radio interferometría y síntesis de imágenes en astronomía
Departamento de Ingeniería en Informática
Proyecto - Parte 2

1 Objetivo

El objetivo de este laboratorio es implementar algoritmos de optimización para la reconstrucción de imágenes astronómicas a partir de visibilidades grideadas, optimizando el proceso tanto para CPU como para GPU. Se busca acelerar el proceso de gridding mediante el uso de CuPy y kernels CUDA de Numba, y posteriormente aplicar algoritmos de optimización eficientes para GPU con el fin de reconstruir la imagen I desde las visibilidades grideadas obtenidas en el laboratorio anterior.

2 Reconstrucción de imágenes mediante optimización

En el laboratorio anterior se construyó la *dirty image* a partir de las visibilidades grideadas mediante una transformada inversa de Fourier. Sin embargo, esta imagen contiene artefactos debido al muestreo incompleto del plano uv y a los efectos de la función de gridding. Para obtener una imagen más precisa, es necesario resolver un problema de optimización inversa.

La relación entre la imagen I y las visibilidades V puede expresarse como:

$$V = \mathcal{F}(I) + \epsilon \tag{1}$$

donde $\mathcal{F}$ representa el operador de transformada de Fourier (considerando el gridding) y ϵ representa el ruido observacional.

Para reconstruir la imagen I, se busca minimizar una función objetivo que combina un término de fidelidad a los datos y un término de regularización:

$$\hat{I} = \arg\min_I \left\{ \frac{1}{2}\|V - \mathcal{F}(I)\|_W^2 + \lambda R(I) \right\} \tag{2}$$

donde $\|\cdot\|_W^2$ representa la norma ponderada por los pesos W de las visibilidades (también conocido como χ^2 o chi cuadrado), λ es el parámetro de regularización y $R(I)$ es el término de regularización que promueve soluciones con propiedades deseables (por ejemplo, suavidad o sparsity).

El primer término de la ecuación 2, conocido como χ^2 o chi cuadrado, mide qué tan bien la imagen reconstruida se ajusta a las visibilidades observadas. Si derivamos este término con respecto a la imagen I, obtenemos que el gradiente del término de verosimilitud es:

$$\nabla_I \left(\frac{1}{2}\|V - \mathcal{F}(I)\|_W^2 \right) = \mathcal{F}^* \left(W \odot (V - \mathcal{F}(I)) \right) \tag{3}$$

donde $\mathcal{F}^*$ es el operador adjunto (transformada inversa de Fourier considerando el gridding), $\odot$ denota el producto elemento a elemento y W son los pesos de las visibilidades. Esto significa que el gradiente del término de fidelidad es simplemente la transformada inversa de Fourier de las visibilidades grideadas residuo (diferencia entre las visibilidades observadas y las predichas), ponderadas por los pesos.

3 Parte 1: Optimización del gridding con GPU

Antes de proceder con la reconstrucción mediante optimización, es necesario optimizar el proceso de gridding para acelerar los cálculos. Esta optimización es crucial ya que el proceso de optimización requerirá múltiples evaluaciones del operador directo e inverso.

3.1 Implementación con CuPy

Implemente una versión del algoritmo de gridding usando CuPy para operaciones en GPU. Las operaciones básicas de gridding deben realizarse completamente en GPU, evitando transferencias innecesarias entre CPU y GPU.

3.2 Kernels CUDA con Numba

Complemente la implementación con CuPy utilizando kernels CUDA personalizados escritos con Numba. Estos kernels deben optimizar las operaciones más costosas del proceso de gridding, tales como:

- El cálculo de índices de grilla a partir de coordenadas uv
- La acumulación de visibilidades en la grilla
- La normalización por pesos acumulados

3.3 Agregado de ruido gaussiano

Antes de realizar el gridding, agregue ruido gaussiano a las visibilidades simuladas. El ruido debe agregarse de forma independiente tanto a la parte real como a la parte imaginaria de cada visibilidad $V(u, v)$:

$$V_{\text{ruidosa}}(u, v) = V(u, v) + \epsilon_{\text{real}} + j\epsilon_{\text{imag}} \tag{4}$$

donde ϵ_{real} y ϵ_{imag} son variables aleatorias gaussianas independientes con media cero y desviación estándar σ. El nivel de ruido debe ser seleccionado de manera que proporcione un ratio señal-ruido (SNR) razonable para el problema de reconstrucción (sugerencia: usar σ tal que el SNR esté entre 10 y 30 dB).

Esta adición de ruido simula las condiciones reales de observación radio-interferométrica, donde las medidas están afectadas por ruido térmico y otros efectos observacionales. El proceso de optimización deberá ser robusto a este ruido para lograr una reconstrucción adecuada de la imagen.

3.4 Comparación de rendimiento

Evalúe el rendimiento comparando:

- Implementación original en CPU (numpy)
- Implementación optimizada con CuPy
- Implementación optimizada con kernels CUDA de Numba

Reporte los tiempos de ejecución para diferentes tamaños de grilla y números de visibilidades, y analice las ganancias de velocidad obtenidas.

4 Parte 2: Reconstrucción mediante optimización

A partir de las visibilidades grideadas obtenidas en el laboratorio anterior, implemente algoritmos de optimización para reconstruir la imagen I. Los algoritmos deben estar optimizados para operar completamente en GPU.

4.1 Formulación del problema

Defina claramente:

- El operador directo $\mathcal{F}$ que mapea la imagen al espacio de Fourier (considerando el gridding)
- El operador adjunto $\mathcal{F}^*$ que mapea las visibilidades grideadas de vuelta a la imagen
- La función objetivo y su gradiente
- El término de regularización $R(I)$ con su función, gradiente y operador proximal (ver sección 4.2)

4.2 Regularización

El término de regularización $R(I)$ es fundamental para obtener imágenes reconstruidas de buena calidad, especialmente cuando los datos están incompletos o ruidosos. Debe implementar las siguientes opciones de regularización, cada una con su función, gradiente y operador proximal.

4.2.1 Operador proximal

El *operador proximal* de una función R es una herramienta fundamental en optimización convexa. De manera simple, el operador proximal $\text{prox}_{\lambda R}(x)$ resuelve el problema de minimizar una función R mientras se mantiene cerca de un punto dado x. Matemáticamente se define como:

$$\text{prox}_{\lambda R}(x) = \arg\min_y \left\{ R(y) + \frac{1}{2\lambda}\|y - x\|^2 \right\} \tag{5}$$

donde λ controla qué tan cerca queremos estar del punto x. En términos simples, el operador proximal encuentra el punto que minimiza R pero penaliza alejarse demasiado de x. Algunos algoritmos como MFISTA requieren explícitamente el operador proximal en lugar del gradiente.

4.2.2 Norma L1

La norma L1 promueve soluciones sparse (con muchos valores cero o cercanos a cero), lo cual es útil para reconstruir imágenes con fuentes puntuales o estructuras localizadas. Esta regularización puede utilizarse con todos los algoritmos de optimización mencionados (Gradiente Conjugado, LBFGS y MFISTA).

Función:

$$R_{\mathrm{L1}}(I) = \|I\|_1 = \sum_{i,j} |I_{i,j}| \tag{6}$$

Gradiente:

$$\nabla R_{\mathrm{L1}}(I) = \mathrm{sign}(I) \tag{7}$$

donde $\mathrm{sign}(I)$ aplica la función signo elemento a elemento (retorna $+1$ si $I_{i,j} > 0$, -1 si $I_{i,j} < 0$, y 0 si $I_{i,j} = 0$).

Proximal:

$$\mathrm{prox}_{\lambda R_{\mathrm{L1}}}(x) = \text{soft-threshold}(x, \lambda) = \mathrm{sign}(x) \odot \max(|x| - \lambda, 0) \tag{8}$$

donde $\odot$ denota producto elemento a elemento. Este operador se conoce como *soft-thresholding* y contrae los valores hacia cero.

4.2.3 Total Squared Variation (TSV)

El Total Squared Variation promueve suavidad en la imagen al penalizar variaciones grandes entre píxeles vecinos, siendo útil para imágenes con regiones uniformes.

Función:

$$R_{\mathrm{TSV}}(I) = \sum_{i,j} \left[(I_{i+1,j} - I_{i,j})^2 + (I_{i,j+1} - I_{i,j})^2 \right] \tag{9}$$

Gradiente: El gradiente de TSV requiere calcular diferencias finitas en ambas direcciones espaciales:

$$\nabla R_{\mathrm{TSV}}(I) = -2 \left[(I_{i+1,j} - I_{i,j}) - (I_{i,j} - I_{i-1,j}) + (I_{i,j+1} - I_{i,j}) - (I_{i,j} - I_{i,j-1}) \right] \tag{10}$$

que puede expresarse de forma más compacta usando el operador Laplaciano discreto.

Proximal: Para TSV, el operador proximal puede resolverse usando métodos iterativos como el algoritmo de Chambolle, o aproximarse mediante métodos de gradiente conjugado para resolver el sistema lineal resultante.

4.2.4 Entropía

La regularización por entropía promueve distribuciones suaves y positivas, siendo especialmente útil cuando se requiere que la imagen sea no negativa y tenga propiedades estadísticas específicas.

Función:

$$R_{\mathrm{Ent}}(I) = \sum_{i,j} I_{i,j} \log(I_{i,j} + \epsilon) - I_{i,j} \tag{11}$$

donde ϵ es un pequeño valor positivo para evitar problemas numéricos cuando $I_{i,j} = 0$.

Gradiente:

$$\nabla R_{\mathrm{Ent}}(I) = \log(I + \epsilon) \tag{12}$$

Proximal: El operador proximal de la entropía puede resolverse usando el método de Newton o aproximaciones iterativas, ya que requiere resolver:

$$\mathrm{prox}_{\lambda R_{\mathrm{Ent}}}(x) = \arg\min_y \left\{ y \log(y + \epsilon) - y + \frac{1}{2\lambda}(y - x)^2 \right\} \tag{13}$$

4.3 Algoritmos de optimización

Importante: Los algoritmos de optimización deben ser implementados desde cero. No está permitido el uso de librerías externas como SciPy, scipy.optimize, o cualquier otra librería que contenga implementaciones pre-hechas de estos algoritmos. Sin embargo, sí se permite utilizar funciones básicas de NumPy/CuPy para operaciones elementales (transformadas de Fourier, operaciones matemáticas básicas, etc.).

Implemente y compare los siguientes algoritmos de optimización, todos optimizados para GPU:

1. **Gradiente conjugado no lineal (Non-linear Conjugate Gradient)**: Implemente el algoritmo de gradiente conjugado de Fletcher-Reeves o Polak-Ribière, adaptado para problemas no lineales.

2. **LBFGS (Limited-memory BFGS)**: Implemente el algoritmo quasi-Newton de memoria limitada LBFGS, que aproxima la matriz Hessiana inversa usando información de gradientes anteriores.

3. **MFISTA (Monotone Fast Iterative Shrinkage-Thresholding Algorithm)**: Implemente el algoritmo MFISTA, una variante del algoritmo FISTA que garantiza monotonía en la función objetivo.

4.4 Selección de algoritmo

De los tres algoritmos implementados, seleccione uno y justifique su elección basándose en:

- Velocidad de convergencia observada
- Calidad de la imagen reconstruida
- Estabilidad numérica
- Eficiencia computacional en GPU

4.5 Validación

Valide la reconstrucción comparando:

- La imagen reconstruida con la imagen original (si está disponible)
- La imagen reconstruida con la dirty image obtenida en el laboratorio anterior
- La convergencia del algoritmo (gráfico de función objetivo vs iteración)
- El tiempo de ejecución total del proceso de reconstrucción

5 Consideraciones técnicas

5.1 Optimización para GPU

Todas las operaciones deben ejecutarse en GPU evitando transferencias innecesarias entre CPU y GPU. Considere:

- Usar arrays de CuPy para todas las operaciones principales
- Minimizar el uso de operaciones que requieran sincronización de GPU
- Optimizar el uso de memoria de GPU para evitar desbordamientos
- Implementar kernels CUDA eficientes cuando sea necesario

5.2 Parámetros de optimización

Documente y justifique la selección de:

- Parámetro de regularización λ
- Tamaño de grilla $N \times N$
- Criterios de convergencia (tolerancia, número máximo de iteraciones)
- Parámetros específicos de cada algoritmo (tamaño de memoria para LBFGS, tamaño de paso para gradiente conjugado, etc.)

6 Entregables

- Un jupyter notebook o Collab notebook con toda la implementación
- Archivos Python adicionales en caso de usar módulos separados
- Comparación de rendimiento del gridding optimizado (tablas y/o gráficos)
- Comparación de los tres algoritmos de optimización implementados
- Justificación de la selección del algoritmo elegido
- Imágenes reconstruidas usando el algoritmo seleccionado
- Análisis de convergencia de los algoritmos

Fecha de entrega (deadline): 5 de diciembre antes de las 23:59 hrs.

.