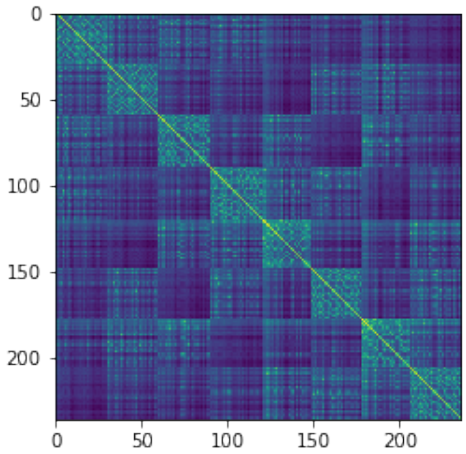
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
IMT3810 - Tópicos Avanzados en Análisis Numérico
Profesor: Elwin van't Wout

Proyecto

Matrices Jerárquicas



Estudiante: Francisca Muñoz

Fecha entrega: 17 de diciembre de 2023

Repositorio en GitHub: `https://github.com/franamr/ProyectoIMT3810`

Contents

1 Introducción y Metodología

Considerando que el método de elementos de frontera "BEM" genera una matriz densa, que necesita mucha memoria para poder guardarse, vamos a desarrollar un método para poder hacer las operaciones de forma más eficiente.

Para esto, utilizando la librería `Bempp-cl` realizaremos la compresión jerárquica de dichas matrices. Esta compresión consiste en descomponer la matriz en diferentes bloques. Algunos de estos bloques serán aproximados por aproximaciones de bajo rango y otras serán mantenidas. Esto es posible por la construcción del método Bem y el uso de la función de Green, ya que los nodos más cercanos entre sí tendrán un peso mayor a los lejanos.

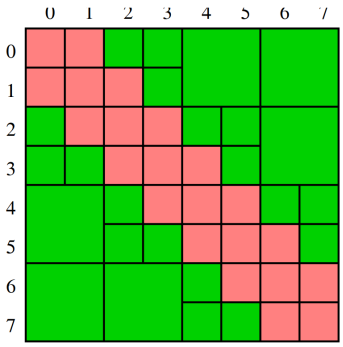


Figure 1: Partición para $p = 3$

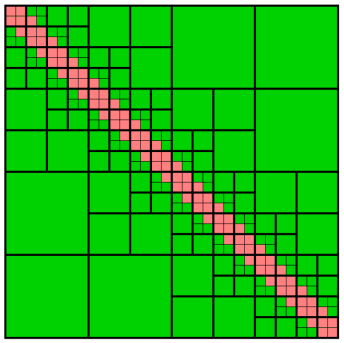


Figure 2: Partición para $p = 5$

La construcción de la matriz jerárquica o $\mathcal{H}$-Matrix se realizará de forma recursiva. Si consideramos que para la primera partición de la matriz $M \in \mathbb{R}^{2^n \times 2^n}$ obtendremos:

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} =: \mathcal{H}_1$$

Con $M_{11}, M_{12}, M_{21}, M_{22} \in \mathbb{R}^{2^{n-1} \times 2^{n-1}}$. Luego a las matrices M_{12} y M_{21} se les hará una aproximación de bajo rango utilizando **ACA**, mientras que en las matrices diagonales continuará el algoritmo recursivo. Así, obtenemos la siguiente representación para un operador de matriz en descomposición jerárquica:

$$\mathcal{H}_p = \begin{bmatrix} \mathcal{H}_{p-1} & \mathcal{R}_{p-1} \\ \mathcal{R}_{p-1} & \mathcal{H}_{p-1} \end{bmatrix}$$

Con $\mathcal{H}_p \in \mathbb{R}^{2^p \times 2^p}$ y $\mathcal{H}_{p-1}, \mathcal{R}_{p-1} \in \mathbb{R}^{2^{p-1} \times 2^{p-1}}$. Considerando que en las matrices $\mathcal{H}$ se continúa con la recursión, y en las matrices $\mathcal{R}$ se realiza la descomposición de bajo rango. Así obtendremos una compresión de la forma mostrada en las Figuras 1 y 2.

Algunos atributos clave, como el número de bloques en los que se va a descomponer la matriz, el número de bloques de bajo rango, el número de bloques denso y la cantidad de memoria utilizada dependerán también de la forma en que se distribuirán los bloques de sub-matrices.Esto se obtiene a partir de la **condición de admisibilidad**, que más adelante será descrita.

Hay ciertos atributos y/o magnitudes que es necesario tener en cuenta a la hora de realizar esta compresión: Podemos ver que por cada fila, tendremos 3 bloques densos, y en la priera y última fila solo serán 2.

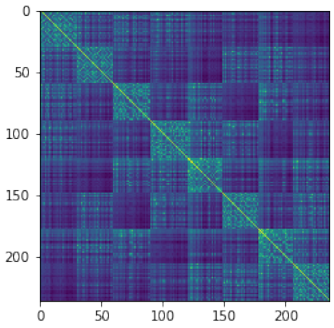


Figure 3: Ejemplo de single layer Potential para Laplace con malla esférica

2 Conceptos importantes

2.1 Adaptative Cross Approximation (ACA)

Las matrices definidas anteriormente como $\mathcal{R}_p$, serán aproximadas por el método de bajo rango "Adaptive cross Approximation". En este método el rango de la aproximación será determinado en función de la submatriz, para disminuir el error convenientemente. Este método permite descomponer la matriz en el producto de dos matrices rectangulares, disminuyendo considerablemente el costo de almacenamiento.

A continuación se presenta un pseudocódigo del algoritmo para matrices cuadradas. El

Algorithm 1: Adaptive Cross Approximation

Data: $M \in \mathbb{R}^{\hat{t}\times\hat{s}}$, $\tau > 0$ (determina precisión), $\varepsilon_{abs} > \tau$
Result: Aproximación de rango k variable $\sum_{v=1}^{k} a^v (b^v)^T$

while $\varepsilon_{abs} > \tau$ **do**
  Se busca la entrada en módulo más grande;
  $(i_v, j_v) := \text{argmax}_{(i,j)} |M_{i,j}|, \qquad \delta := |M_{i_v,j_v}|;$
  if $\delta = 0$ **then**
    El algoritmo termina con rano $v - 1$;
  else
    Se computan las entradas de a^v y b^v ;
    $(a^v)_i := M_{i,j_v}, \quad i \in \hat{t}, \qquad (b^v)_j := M_{i_v,j}/\delta, j \in \hat{s}$;
    Se actualizan los valores de la matriz ;
    $M_{i,j} := M_{i,j} - (a^v)_i (b^v)_j, \quad i \in \hat{t}, j \in \hat{s};$
    $\varepsilon_{abs} = \|a^v\|_2 \|b^v\|_2;$
    $v = v + 1$
return a^v, b^v

pseudocódigo anterior, se encuentra en el repositorio entregado. Así, por cada sub-matriz, que cumpla con el criterio de admisibilidad, será descompuesta en el producto de dos matrices rectangulares más pequeñas.

2.2 Condición de Admisibilidad (Admissibility condition)

Como se mencionó anteriormente, la condición de admisibilidad nos indicará de que manera se deberá realizar la partición de la matriz original.
A continuación se presentan dos ejemplos clásicos de dichas particiones.

La partición utilizada depende también de la dimensionalidad del problema, entre otros aspectos.
En las imágenes anteriores vemos dos criterios de partición diferentes, llamados **Condición de admisibilidad débil** y **Condición de admisibilidad fuerte**, respectivamente. En el primer caso (Figura 4) se ve el criterio de admisibilidad débil, en que únicamente los bloques diagonales se almacenarán como bloques densos, mientas que para todos los otros bloques se realizará una aproximación de bajo rango (ACA). Esta aproximación se realiza de manera recursiva, sobre los bloques que no forman parte de la diagonal de la matriz, quedando así finalmente los bloques diagonales como densos. Por otro lado, la condición de admisibilidad fuerte (Figura 5) es similar a la débil, pero presentando una mayor can-

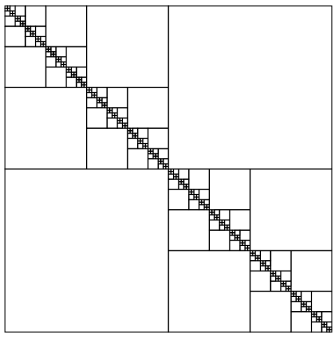


Figure 4: Weakly admissibility condition

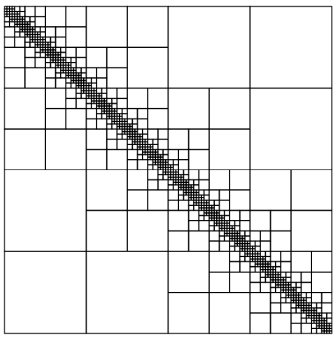


Figure 5: Strongly admisibility condition

tidad de sub-matrices, como se puede apreciar en la figura. En este caso, las matrices densas son las correspondientes a la diagonal y también sus vecinas (Se observa en Figura 1 y Figura 2 igualmente).

Es importante destacar que otro tipo de operadores, por ejemplo para tratar problemas de Helmholtz, o Maxwell o también con otro tipo de geometrías, se deberá utilizar otro criterio de admisibilidad. Al ser estos problemas de mayor complejidad, se deberán calcular las distancias dentro de cada bloque y con los bloques vecinos, para determinar qué valores podrán ser aproximados. Esto teniendo en cuenta los nodos de la malla y las interacciones cercanas y lejanas entre ellos. No se profundizará mayormente en el criterio matemático de admisibilidad, ya que en el código entregado solo se utiliza la condición débil.

2.3 Operaciones: Matvec

Para la implementación de la operación **matriz** $\times$ **vector**, se utilizó el caso específico para la condición débil de admisibilidad. Para esto, la función debe ser definida de forma recursiva sobre los bloques de sub-matrices. A continuación se encuentra la forma de implementación, considerando la siguiente notación: $(M_{i,j})^k$, i : fila, j : columna, k : número de iteración.

$$\begin{pmatrix} M^1_{1,1} & M^1_{1,2} \\ M^1_{2,1} & M^1_{2,2} \end{pmatrix} \cdot \begin{pmatrix} b^1_1 \\ b^1_2 \end{pmatrix} = \begin{pmatrix} M^1_{1,1} b^1_1 + M^1_{1,2} b^1_2 \\ M^1_{2,1} b^1_1 + M^1_{2,2} b^1_2 \end{pmatrix}$$

Luego, las operaciones $M^1_{1,1} b^1_1$ y $M^1_{2,2} b^1_2$ deben realizarse de forma recursiva hasta llegar a las sub-matrices densas. Por otro lado, las operaciones $M^1_{1,2} b^1_2$ y $M^1_{2,1} b^1_1$ se realizarán, descomponiendo las matrices según la Adaptive Cross Approximation. Así obtenemos :

$$M^1_{1,2} b^1_2 \simeq A^1_{2,1} (B^{1T}_{1,2} b^1_2)$$

$$M^1_{1,2} b^1_2 \simeq A^1_{1,2} (B^{1T}_{1,2} b^1_2)$$

Finalmente, realizando este procedimiento de forma recursiva, obtenemos el resultado para matrices con condición de admisibilidad débil. El código de este procedimiento se encuentra en el GitHub.
En el siguiente algoritmo para la operación Matvec en matrices jerárquicas, se considera la función `ACA_matvec` que realiza el producto matvec conuna matriz que presente descomposición ACA.

Algorithm 2: Hmatvec, matrices jerárquicas (recursivo)

Data: $M \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, k: dimensión de las sub-matrices densas.
Result: Vector $c \in \mathbb{R}^n$ tal que $Mb = c$

if $n < k$ **then**
    retornar Mb (producto matvec clásico)
else
    Aplicamos recursivamente el algoritmo a cada bloque $M_{11} = M[: n/2, : n/2]$
    $M_{12} = M[: n/2, n/2 :]$
    $M_{21} = M[n/2 :, : n/2]$
    $M_{22} = M[n/2 :, n/2 :]$
    $b_1 = b[: n/2]$
    $b_2 = b[n/2 :]$
    $c1 = hmatvec(M_{11}, b_1) + ACA\_matvec(M_{12}, b_2)$
    $c2 = hmatvec(M_{21}, b_1) + ACA\_matvec(M_{22}, b_2)$
return c

3 Ejemplo: Single Layer Potential para formulación de Laplace

Para comenzar, realizaremos la compresión en matriz jerárquica de el operador **"Single Layer potential"** para Laplace, ya que es el más simple. En el notebook incluido en el repositorio de GitHub se encuentra un ejemplo realizado para una malla esférica y utilizando funciones de base **P1**.

Segun el texto *Hierarchical Matrices Based on a Weak Admissibility Criterion*(W. Hackbusch, B. N. Khoromskij, R. Kriemann), la condición débil de admisibilidad es apropiado para utilizar con el potencial de Single-Layer para el problema de Laplace.

Para esto se siguió la siguente metodología:

1. Crear el problema utilizando la formulación de Laplace.
2. Construir el Sinlge Layer Potential utilizando Bempp.
3. Se realiza la partición en bloques según la condición de admisibilidad débil.
4. Los criterios correspondientes de comprimen utilizando ACA.
5. Para la operación MatVec, se realiza otra función recursiva.

Todo el código se encuentra en GitHub.

4 Conclusiones

A partir de los aspectos teóricos comentados anteriormente, se logró crear funciones de compresión jerárquica, de descomposición en ACA y de multiplicación matriz por vector para compresión jerárquica. Sin embargo estos resultados se obtuvieron solo para un tipo de matriz muy específico (single Layer potential de Laplace). Es ideal generar funciones que permitan la compresión de todo tipo de operadores y también para todo tipo de matrices en general. Además es necesario agregar más operaciones, como suma de matrices o matmat. Por último para agregar la funcionalidad completa de matrices jerárquicas a Bempp también es útil agregar los parámetros mencionados al inicio de este informe, como cantidad de bloques en que se descompone la matriz, cantidad de bloques densos y cantidad de memoria utilizada. Esto se debe realizar haciendo una implementación más exhaustiva aplicable a todo tipo de matrices.

5 Uso de Herramientas externas

Para este proyecto se utilizaron las librerías Bempp y Numpy en la plataforma VS-Code. Además se utilizaron las extensiones `Flake8` y `Autopep8` para controlar el formato PEP8. Por último en determinadas dudas básicas sobre numpy se consultó con ChatGPT temas sobre formato. Dichas respuestas fueron confirmadas y testeadas, ya que se utilió para el ahorro de tiempo.

6 Referencias

1 Betcke, Timo, Elwin van 't Wout, and Pierre Gélat (2017). "Computationally efficient boundary element methods for high-frequency Helmholtz problems in unbounded domains". In: Modern Solvers for Helmholtz Problems. Ed. by Domenico Lahaye, Jok Tang, and Kees Vuik. Geosystems Mathematics. Cham: Birkh äuser, pp. 215–243. doi: 10.1007/978-3-319-28832-1_9.

2 Börm, Steffen (2010). Efficient numerical methods for non-local operators: H2-matrix compression, algorithms and analysis. Vol. 14. EMS Tracts in Mathematics. Zürich: European Mathematical Society.

3 Börm, Steffen, Lars Grasedyck, and Wolfgang Hackbusch (2003). Hierarchical Matrices. Tech. rep. Leipzig: Max-Planck-Institut für Mathematik in den Naturwissenschaften. https://www.mis.mpg.de/publications/other-series/ln/lecturenote-2103.html

4. W. Hackbusch, B. N. Khoromskij, R. Kriemann (2004) "Hierarchical Matrices Based on a Weak Admissibility Criterion", https://link.springer.com/content/pdf/10.1007/s00607-004-0080-4.pdf