**Solvers directos vs solvers iterativos en Matlab.**

Preparado por: C. Jorge Caballero Rosales.

Coordinado por: Arturo Benjamín Hurtado Pérez y

Jorge Javier Hernández Gómez.

25 de febrero del 2022.

**Métodos Directos vs Métodos Iterativos.**

**Sistemas lineales A\*x=b.**

Una de las aplicaciones más importantes y comunes del álgebra lineal numérica es la solución de sistemas lineales que se pueden expresar de la forma **A\*x = b**. Cuando A es una matriz dispersa grande, existen 2 tipos de métodos para resolver este tipo de sistemas: los métodos directos y los métodos iterativos.

1. Métodos Directos.

Los métodos directos son variantes de la eliminación Gaussiana. Estos métodos utilizan los elementos individuales de las matrices con las operaciones LU, QR y la factorización de Cholesky. La principal ventaja de utilizar métodos directos es su alto nivel de precisión, sin embargo, estos métodos tardan al operar con matrices dispersas largas. La velocidad para resolver sistemas lineales, por medio de métodos directos, depende de la densidad y el patrón de llenado de los coeficientes de la matriz

1. Factorización LU:

Expresa cualquier matriz cuadrada A como el producto de una permutación de una matriz triangular inferior y una matriz triangular superior.

**A= LU**

Donde:  
L= Una permutación de una matriz triangular inferior con unos en su diagonal.   
U= Una matriz triangular superior.

* + 1. Factorización QR

Expresa el producto de una matriz ortogonal o unitaria con una matriz triangular superior.

**A= QR**

Donde:   
Q = matriz ortogonal o unitaria.  
R = Matriz superior triangular.

* + 1. Factorización de Cholesky

Expresa una matriz simétrica como el producto de una matriz triangular y su matriz traspuesta.

**A= R´R**

Donde:   
R= una matriz triangular superior.

1. Métodos Iterativos.

Estos métodos son útiles para sistemas de ecuaciones largos donde es razonable intercambiar la precisión de la solución por un tiempo de ejecución más corto. Los métodos iterativos únicamente utilizan indirectamente la matriz de coeficientes, a través de un producto matriz-vector. La velocidad para resolver sistemas lineales con métodos indirectos no depende del patrón de llenado de la matriz de coeficientes. Sin embargo, para utilizar un método iterativo se requiere ajustar los parámetros para cada problema específico.

Algunos ejemplos de métodos iterativos son:

* + 1. Método de Jacobi

El método de Jacobi es un método iterativo, usado para resolver sistemas de ecuaciones lineales del tipo. El método de Jacobi consiste en usar fórmulas como iteración de punto fijo. La base del método consiste en construir una sucesión convergente definida iterativamente. El límite de esta sucesión es precisamente la solución del sistema. A efectos prácticos si el algoritmo se detiene después de un número finito de pasos se llega a una aproximación al valor de x de la solución del sistema.

* + 1. Método de Gauss-Seidel

Aunque este método puede aplicarse a cualquier sistema de ecuaciones lineales que produzca una matriz (cuadrada, naturalmente pues para que exista solución única, el sistema debe tener tantas ecuaciones como incógnitas) de coeficientes con los elementos de su diagonal no-nulos, la convergencia del método solo se garantiza si la matriz es diagonalmente dominante o si es simétrica y, a la vez, definida positiva.

* + 1. Método SOR

El método de sobre-relajación sucesiva (SOR), es una variante del método de Gauss-Seidel para estimar la solución de un sistema lineal de ecuaciones, permitiendo una convergencia más rápida.

* + 1. Método del Gradiente conjugado.

El método del gradiente conjugado es un algoritmo para resolver numéricamente los sistemas de ecuaciones lineales cuyas matrices son simétricas y definidas positivas.

1. Método de Elementos Finitos FEM.

El método de elementos finitos es un al algoritmo que utilizan en softwares de simulación computacional para poder encontrar la solución a diversos problemas utilizando una división geométrica, es decir divide el sistema en subsistemas más simples para poder resolverlos individualmente.

En la rama de la ingeniería mecánica la mayoría de los problemas se representan por la ecuación del equilibrio mecánico KU=F, que en la mayoría de los casos viene como un sistema de ecuaciones lineales, el cual puede ser resuelto bajo el método de elementos finitos, Matlab es uno de los softwares de simulación que se encargan de resolver dichos sistemas bajo este método.

**¿Cuándo utilizar un solver directo en el ámbito FEM?**

En Matlab los solvers directos son los algoritmos de solución basados en los métodos directos vistos anteriormente. Matlab los implementa gracias a los comandos \ o /, sin embargo, estos métodos son variantes de la eliminación gaussiana por lo que, al trabajar con matrices dispersas o sistemas muy grandes, la computadora donde se realiza la simulación demanda demasiados recursos para obtener la solución, además del tiempo que requiere. A pesar de lo anterior mencionado, estos métodos obtienen la solución con un alto grado de precisión lo que significa que, si requerimos una solución exacta del sistema, y nuestra computadora cuenta con los requisitos suficientes, podemos utilizar estos métodos para obtener la solución.

**¿Cuándo utilizar un solver iterativo en el ámbito FEM?**

Por otro lado, Matlab nos da otra opción para obtener la solución de dichos sistemas de matrices dispersas. La implementación de métodos iterativos en Matlab es una de las opciones para resolver sistemas de matrices dispersas o muy grandes y cuando el tiempo de ejecución de un solver directo es muy alto. Estos métodos iterativos se encargan de sacrificar la precisión de la solución del sistema, por la reducción de recursos y el tiempo en el que dan la respuesta. Por lo que estos métodos deben utilizarse cuando no requerimos un alto grado de precisión y/o nuestra computadora no cuenta con suficientes recursos para llevarlo a cabo por un método directo.

Sin embargo, el método iterativo implementado en Matlab depende de las condiciones del sistema, en específico de la matriz de coeficientes que define al sistema. Cada uno de los métodos iterativos cuenta con sus condiciones para poder ser utilizado.

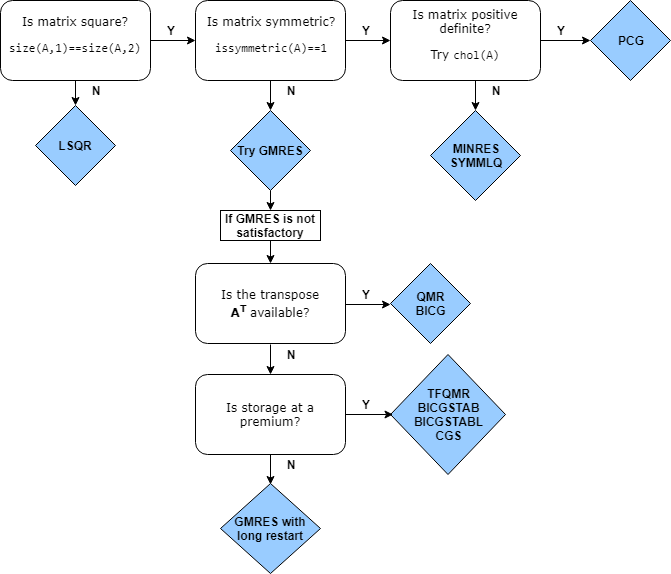
**Solvers iterativos en Matlab.**

1. PCG: Método precondicionado del gradiente conjugado.
2. lsqr: Mínimos cuadrados.
3. minres: Mínimo residual.
4. symmlq: LQ simétrico.
5. bicg: gradiente biconjugado.
6. bicgstab: Gradiente biconjugado estabilizado.
7. bicgstabl: Gradiente biconjugado estabilizado (l).
8. cgs: gradiente conjugado cuadrado.
9. gmres: Residuo mínimo generalizado.
10. qmr: residual cuasi-minimo
11. tfqmr: Residuo cuasi-minimo libre de transposición.

**¿Como implementar un solver iterativo en Matlab?**

En los casos básicos de sistemas lineales la matriz representada por la variable K es una matriz cuadrada, semidefinida positiva, diagonalmente dominante y esparcida. Con las características anteriores debemos encontrar el método iterativo que puede ser utilizado en este caso.

En la figura 1.1 Se puede observar el proceso que hay que llevar a cabo para poder seleccionar el método que más nos convenga para resolver el sistema de ecuaciones lineales.

****

**Figura 1.1 Diagrama de flujo para seleccionar el solver iterativo en Matlab**

1. Uso de los solvers iterativos.

Los solvers iterativos son la principal propuesta de solución a los sistemas de ecuaciones lineales cuando se trata de sistemas dispersos y que son demasiado grandes para ser tratados como métodos directos. Sin embargo, para poder utilizar algunos solvers iterativos es necesario preacondicionar los sistemas.

1. Preacondicionadores en Matlab.

Un preacondicionador es una matriz que se encarga de transformar el sistema en otro con propiedades más favorables para una solución iterativa. El preacondicionamiento intenta mejorar las propiedades de la matriz de coeficientes para su solución. Generalmente se representa con la letra M.

En un caso ideal el preacondicionador transformaría la matriz de coeficientes A en una matriz identidad por lo que cualquier método iterativo haría la convergencia en una sola iteración. Sin embargo, en la realidad encontrar un buen preacondicionador requiere de compensaciones. La transformación en Matlab se realiza en una de las tres formas siguientes:

1. **Preacondicionamiento por la izquierda:**

El preacondicionador M aparece a la izquierda de la matriz A, es decir:

(*M*−1*A* )*x*=(*M*−1 *b*) .

Los solvers iterativos que utilizan este preacondicionamiento son:

* bicg
* gmres
* qmr

1. **Preacondicionamiento por la derecha:**

El preacondicionador M aparece a la derecha de A:

(*A* *M*−1)(*M* *x*)=*b* .

Los solvers iterativos que utilizan este preacondicionamiento son:

* lsqr
* bicgstab
* bicgstabl
* cgs
* tfqmr

1. **Preacondicionamiento dividido.**

El preacondicionamiento dividido se utiliza para matrices simétricas y garantiza que el sistema transformado siga siendo simétrico. El preacondicionador M= H HT se divide y los factores aparecen en diferentes lados de A.

(*H*−1*A* *H*−*T*)*HTx*=(*H*−1*b*)

Los solvers iterativos que utilizan este preacondicionamiento son:

* pcg
* minres
* symmlq

1. Clasificación de los preacondicionadores.
2. **Factorización incompleta:**

Se llama factorizaciones “incompletas” a aquéllas que no calculan la factorización exacta sino una aproximada, despreciando los elementos que se hacen distintos de cero, pero tienen un valor pequeño. Aunque la factorización es incompleta y sólo aproximada, se puede hacer en mucho menos tiempo y para ciertas finalidades es suficiente.

1. Factorización incompleta LU (ILU en matlab).
2. Factorización incompleta de Cholesky (ichol en Matlab).
3. **Métodos ILU e IC paralelos.**
4. **Matriz identidad o unidad.**
5. **Inversa aproximada dispersa (sparse approximate inverse)**
6. El efecto del reordenamiento y el equilibrio.

Al trabajar con sistemas de matrices dispersas que representen problemas computacionales para obtener la solución, es posible que se necesite preacondicionar el sistema de una manera más eficiente que con las funciones de ILU o ichol por si solas. Es necesario aplicar un preacondicionador de mejor calidad o que sea capaz de minimizar la cantidad de cálculos que se realizan para obtener la solución al sistema. La función de equilibrar o “equilibrio” se utiliza para hacer que la matriz de coeficientes K sea más diagonalmente dominante, lo que resulta en un preacondicionamiento de mejor calidad. Por otro lado, también existe la función del reordenamiento o “reordenar”, que se encarga de minimizar el número de factores distintos a cero en la matriz, lo que reduciría la demanda de memoria a la computadora y disminuye los cálculos subsiguientes.

Aplicar estas funciones junto con la factorización incompleta nos generará un preacondicionamiento bastante más estable y un sistema más sencillo para su resolución, tanto que al aplicar nuestro método iterativo se realizaran menos iteraciones lo que resultara en un aumento en la velocidad de convergencia.

1. Método preacondicionado del gradiente conjugado.

El método del gradiente conjugado es uno de los solvers iterativos más eficientes al trabajar con sistemas de matrices dispersas, simétricas y definidas positivas (SPD). Sin embargo, el método del gradiente conjugado demuestra un comportamiento de convergencia bastante pobre al trabajar con sistemas lineales condicionados, como el que fue aplicado en este reporte más adelante.

Debido a lo anterior, se utilizaron diversas técnicas de preacondicionamiento para mejorar la eficiencia de la solución iterativa.

**Algoritmo del PCG.**

1. Sea X0 una suposición arbitraria.
2. r0 = AX0 – b
3. z0=M^-1 r0
4. P0= z0
5. For *j* =0, 1, 2, ..., *MaxIter*
6. Si se cumple el criterio, sale del bucle.
7. Termina el ciclo for.
8. Implementación del PCG en Matlab para el sistema KU=F.

Para este reporte se llevó a cabo la solución a un sistema KU=F, siendo K una matriz dispersa, cuadrada, simétrica y positiva definida. Se ejecutó el programa por medio de un solver directo, y por medio de un solver iterativo (PCG), utilizando 3000 como número máximo de iteraciones y una tolerancia de 1e-8.

1. Tabla comparativa de los diferentes métodos utilizados.

En la siguiente tabla se enlistan los métodos utilizados para resolver el sistema, cada uno con su consumo de RAM respectivo, el número de iteraciones necesarias para la convergencia y el residuo relativo resultante.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Tiempo | RAM máxima. | No. De iteraciones. | Residual |
| Solver directo (backslash). | 27.0704 | 5.4 GB | - | - |
| PCG sin preacondicionar. | 36.7801 | 1.2 GB | 2868 | 9.8e-09 |
| PCG con ichol. | 54.6646 | 1.3 GB | 1164 | 9.6e-09 |
| PCG con matriz unidad | 43.7007 | 1.0 GB | 2868 | 9.8e-09 |
| PCG con ILU | 45.7105 | 1.1 GB | 1165 | 8.3e-09. |

1. Resultados obtenidos.

**Solver directo.**

Tabla

Descripción generada automáticamente

**PCG sin preacondicionar.**

Interfaz de usuario gráfica, Texto, Tabla

Descripción generada automáticamente con confianza media

**PCG con factorización incompleta de Cholesky (ichol).**

**Interfaz de usuario gráfica, Texto, Tabla

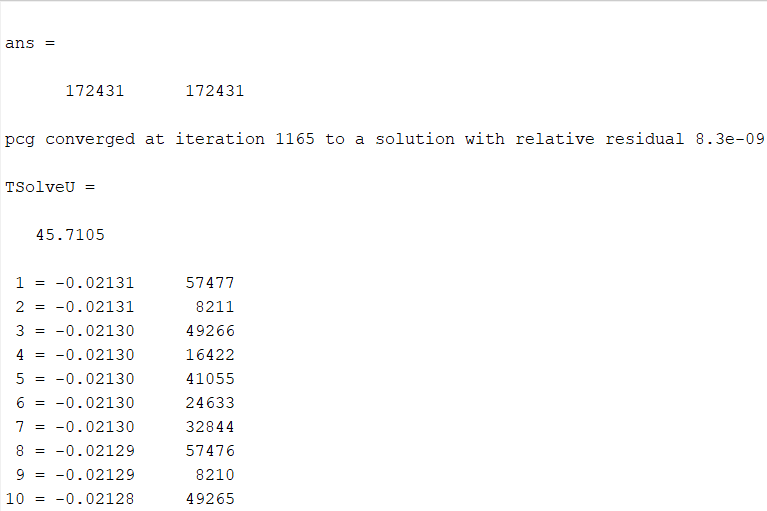
Descripción generada automáticamente con confianza media**

**PCG con matriz unidad.**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**PCG con factorización incompleta LU (ILU).**



1. Código para la solución del sistema en Matlab.

**Codigo del Solver Directo (backslash).**

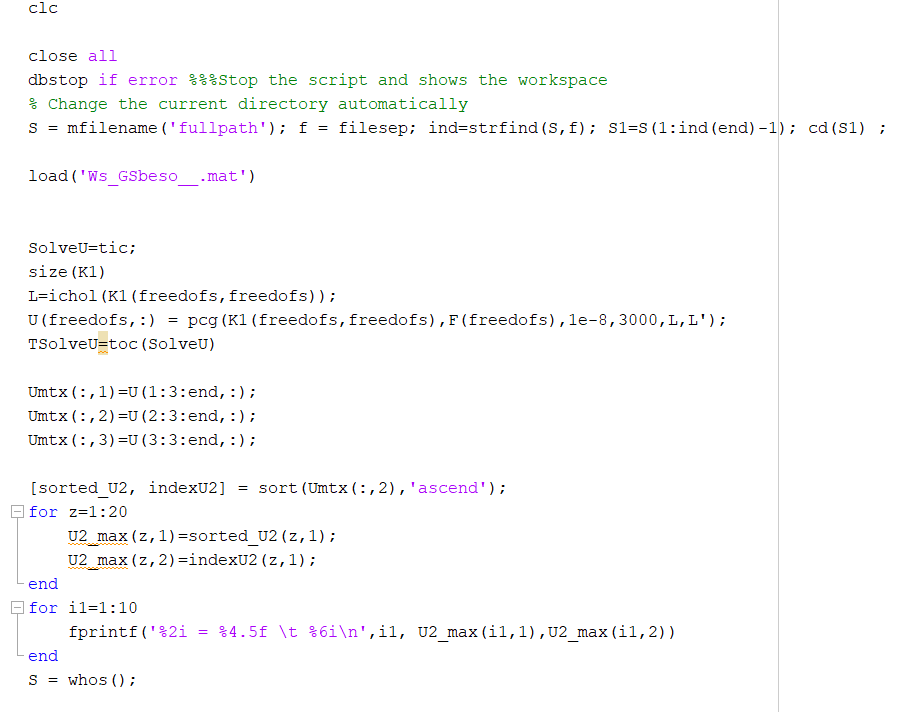
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**PCG sin preacondicionar**



**PCG preacondicionado con ichol.**



**PCG preacondicionado con matriz unidad.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

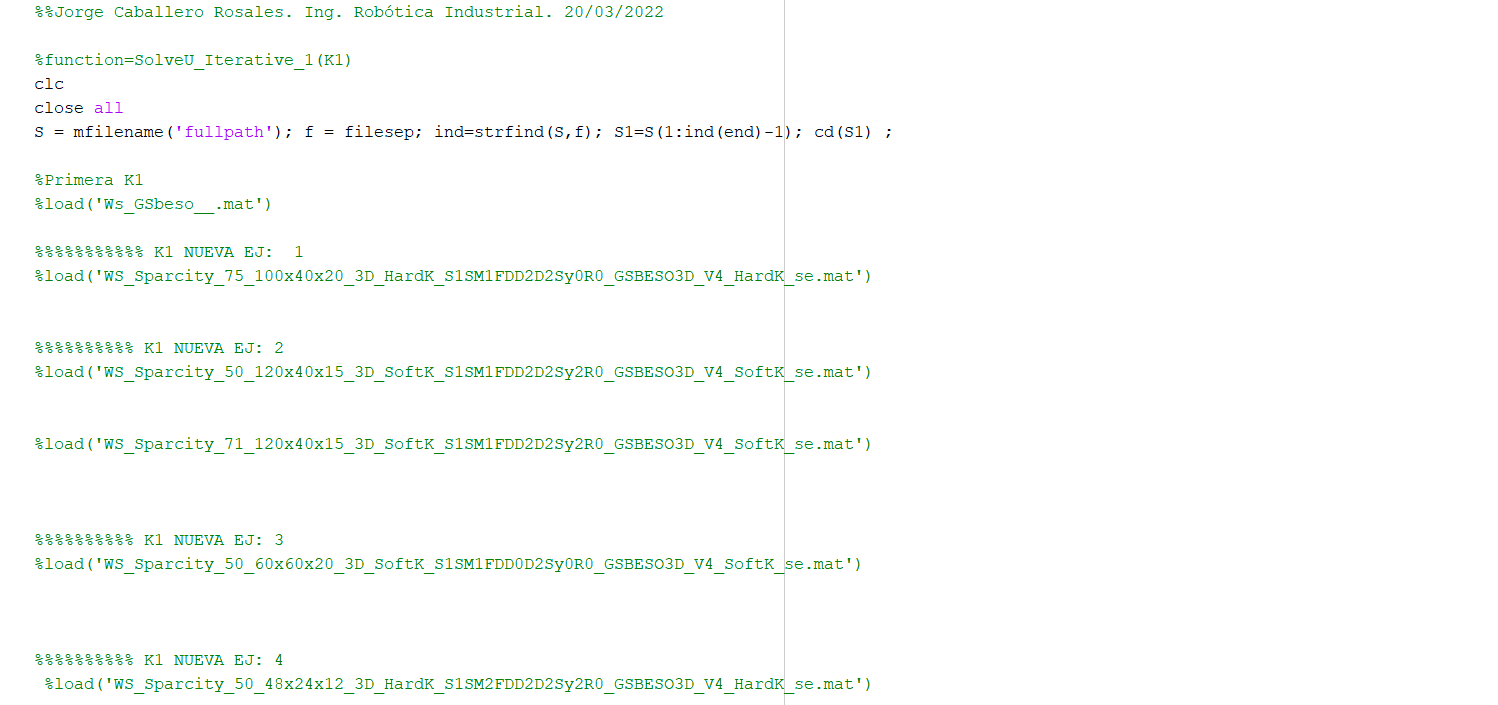
**PCG preacondicionado con ILU.**

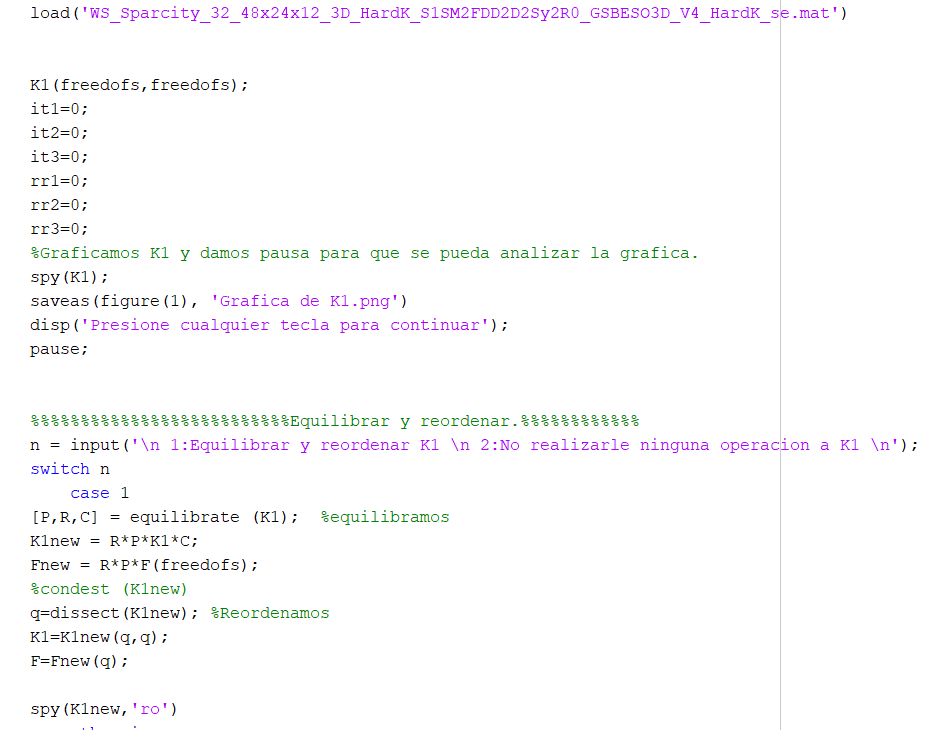
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

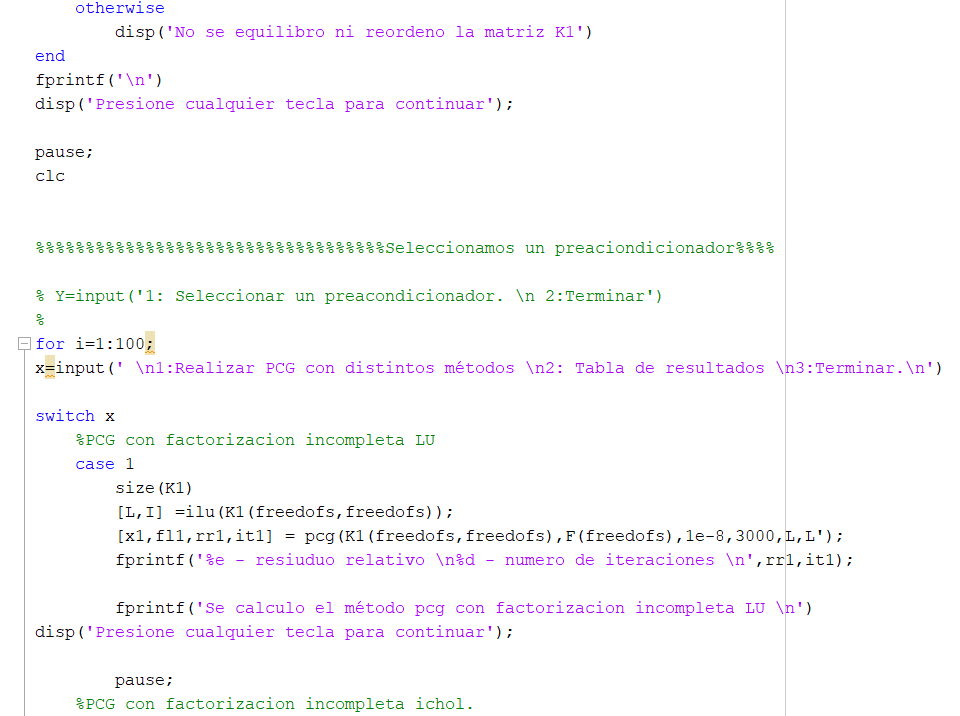
Descripción generada automáticamente

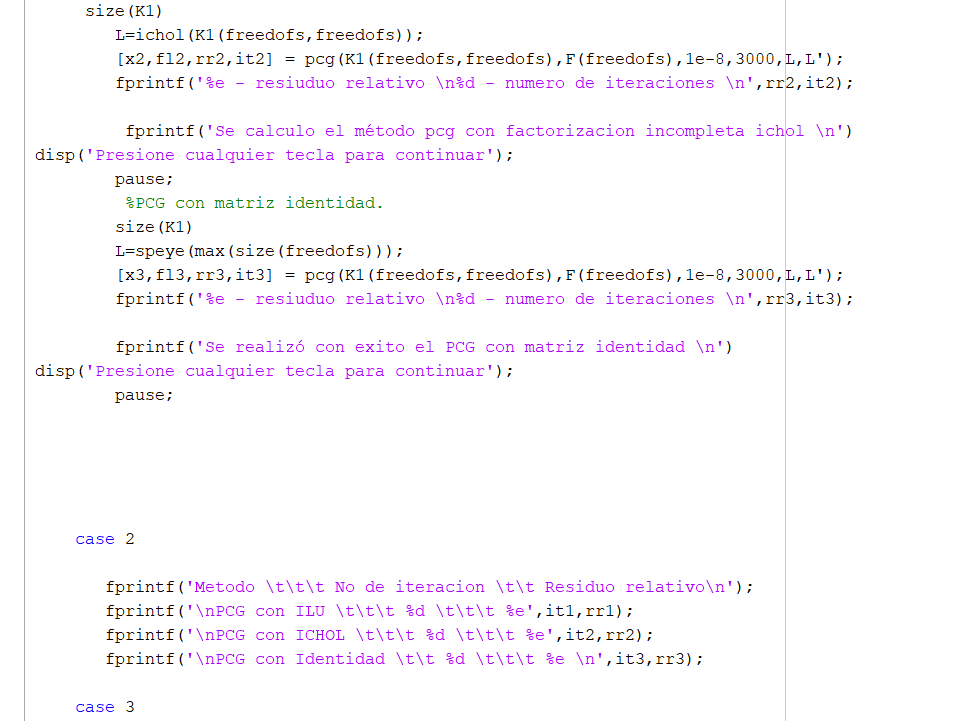
1. Segundo código implementado en Matlab.

Para poder realizar un análisis con mayor profundidad al solver iterativo basado en el método preacondicionado del gradiente conjugado, se implementó un nuevo código en Matlab utilizando distintos preacondicionadores para poder comparar las soluciones obtenidas a distintas matrices dispersas (K1) y encontrar el preacondicionador más apto para solucionar dichos sistemas.





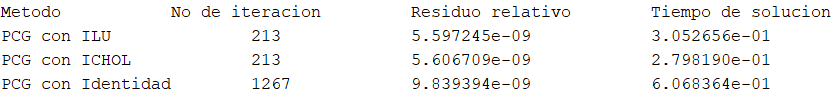




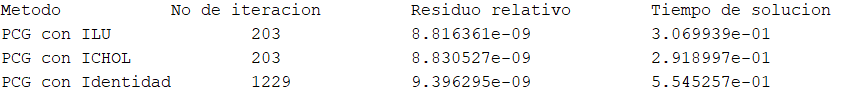


1. Resultados obtenidos.

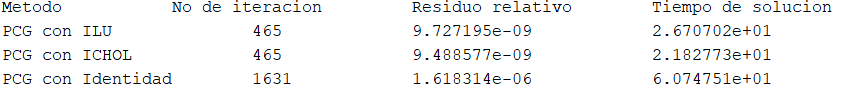
K1 con 47775 grados de libertad.



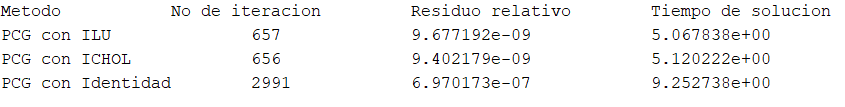
Segunda K1 con 47775 grados de libertad.



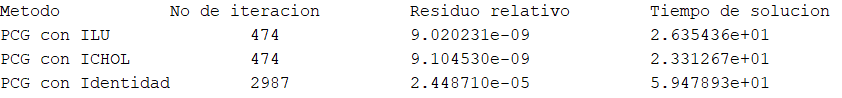
K1 con 234423 grados de libertad.



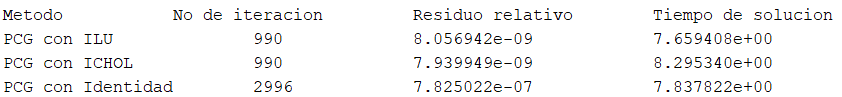
K1 con 238128 grados de libertad.



Segunda K1 con 238128 grados de libertad.



K1 con 260883 grados de libertad.



Este análisis es comparable al estudio realizado por Kardani [KARDANI 2013], en el cual compara distintos preacondicionadores para el método preacondicionado del gradiente conjugado con la finalidad de encontrar la combinación de solver-preacondicionador más eficiente para la solución de estos sistemas.

1. Gráfica de barras comparando el numero de grados de libertad contra el tiempo de solución.
2. Conclusiones del primer código implementado:

Después de realizar la implementación del solver iterativo para nuestro sistema en Matlab, lo primero que resalta es la reducción de demanda a la memoria RAM. Utilizar el solver de PCG con cualquiera de los preacondicionadores reduce considerablemente la demanda de recursos a la computadora con respecto al solver directo. Sin embargo, el tiempo de ejecución para encontrar la convergencia aumenta en todos los casos, pero no en una cantidad considerable.

Durante la solución del sistema se buscaron diferentes métodos para reducir el tiempo necesario para la convergencia, por lo que se utilizó la función de equilibrar. Sin embargo, al visualizar la gráfica del patrón de dispersión de la matriz se puede notar que ya era diagonalmente dominante y que los factores de la matriz distintos a cero no redujeron con cualquiera de las dos funciones para reordenar (symrcm o dissect). Debido a lo anterior las dos funciones no representan una mejoría para este sistema por lo que no fueron consideradas para el cálculo de resultados.

De acuerdo con los resultados obtenidos el método más efectivo para la solución de nuestro sistema es el de PCG preacondicionado con la matriz incompleta LU (ILU), ya que el máximo consumo de memoria RAM es de 1.1GB, que no representa una gran demanda de recursos para cualquier computadora, además de que el tiempo de solución es menor a un minuto por lo que se considera un buen tiempo de respuesta y resulta en un residuo de 8.3e-9, que es el menor de todos los métodos comparados, con una iteración más que el preacondicionador ichol.

1. Conclusiones del segundo código implementado:

Comparando las tablas de resultados obtenidas, podemos notar que la factorización incompleta de cholesky y la factorización incompleta LU requieren casi la misma cantidad de iteraciones para converger y obtienen residuos bastante parecidos, sin embargo, ichol en la mayoría de los casos obtiene un residuo más pequeño con menos iteraciones. Y debido a que existe una relación directa entre el tiempo de solución y las iteraciones necesarias, la función ichol es la que requiere menor tiempo para obtener la solución del sistema por lo que se concluye que este es el preacondicionador más adecuado.

Al analizar los resultados obtenidos de la implementación del segundo código, comparando los distintos preacondicionadores, se concluyó que la combinación solver-preacondicionador más eficiente para resolver estos sistemas es el método del gradiente conjugado preacondicionado con la factorización incompleta de Cholesky, dicha conclusión concuerda con los resultados del estudio realizado por Kardani.

En este código se implementaron las funciones para equilibrar y reordenar la matriz dispersa de coeficientes, sin embargo, por la constitución de las matrices y el algoritmo que requieren dichas funciones, no se logran realizar. No obstante, los métodos comparados no requieren de muchas iteraciones para converger, por lo que este error no es considerable para nuestro análisis.

De la grafica de barras realizada podemos concluir que existe una correlación entre el numero de grados de libertad que tiene nuestra matriz de coeficientes (K1) y el tiempo de solución, siendo esta relación que mientras más alto sea el numero de grados de libertad, mayor será el tiempo requerido para obtener la solución del sistema. Sin embargo, existen otros factores (como la dispersión de la matriz) que afectan el tiempo de solución del sistema.

**Bibliografía.**

Mathworks. (2022). Itherative Methods for Linear Systems. <https://la.mathworks.com/help/matlab/math/iterative-methods-for-linear-systems.html#mw_940f70c5-e98f-4cd0-90b9-3980208a5c3c>

Mathworks. (2022). Preaconditioned conjugate gradients method. <https://la.mathworks.com/help/matlab/ref/pcg.html>

Geisel Y. Alpízar B.(2013) Factorización incompleta de Cholesky como técnica de precondicionamiento. Revista Digital matemática. Vol. 13. No. 1. <http://funes.uniandes.edu.co/8037/1/Alpizar2013Factorizacion.pdf>

Burgos D. A. (2016) Métodos Iterativos para sistemas de ecuaciones lineales. Universidad del Bío-Bío. Facultad de Educación y Humanidades Escuela de Pedagogía en Educación Matemática. <http://repobib.ubiobio.cl/jspui/bitstream/123456789/1228/1/Burgos_Alarcon_Dario.pdf>

Benzi M. (2002) Preaconditioning Techniques for large linear systems: A survey. Journal of Computational Physics 182, (418-477).

Kardani O. et al. (2013) A comparative Study of Preaconditioning Techniques for Large Sparse Systems Arising in Finite Element Limit Analysis. International Journal of Applied Mathematics. 43:4.